Using the Language of Thought

by

Eyal Dechter

A.B., Harvard University (2009)

Submitted to the Department of Brain and Cognitive Sciences
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2018

Author . . . .
Department of Brain and Cognitive Sciences
May 23, 2017

Certified by . . . .
Joshua B. Tenenbaum
Professor
Thesis Supervisor

Accepted by . . . .
Matthew Wilson
Sherman Fairchild Professor of Neuroscience and Picower Scholar,
Director of Graduate Education for Brain and Cognitive Sciences

Using the Language of Thought

by

Eyal Dechter

Submitted to the Department of Brain and Cognitive Sciences
on May 23, 2017, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In this thesis, I develop and explore two novel models of how humans might be able to acquire high-level conceptual knowledge by performing probabilistic inference over a *language of thought* (Fodor 1975) – a space of symbolic and compositional mental representations sufficiently expressive to capture the meanings of human thoughts and utterances. These models and their associated learning algorithms are motivated by an attempt to provide an understanding of the algorithmic principles that might underlie a child's ability to search the haystack of sentences in her language of thought to find the needle that corresponds to any specific concept. The first model takes advantage of the compositionality inherent to LOT representations, framing concept acquisition as program induction in a functional programming language; the *Exploration-Compression* algorithm this model motivates iteratively builds a library of useful program fragments that, when composed, restructures the search space, making more useful programs shorter and easier to find. The second model, the *Infinite Knowledge Base Model* (IKM), frames concept learning as probabilistic inference over the space of relational knowledge bases; the algorithm I develop for learning in this model frames this inference problem as a state-space search over abductive proofs of the learner's observed data. This framing allows us to take advantage of powerful techniques from the heuristic search and classical planning literature to guide the learner. In the final part of this thesis, I explore the behavior of the IKM on several case studies of intuitive theories from the concept learning literature, and I discuss evidence for and against it with respect to other approaches to LOT models.

Thesis Supervisor: Joshua B. Tenenbaum
Title: Professor

Acknowledgments

I would like to thank my advisor, Josh Tenenbaum, for his guidance and support of my work. I am grateful also to the members of my committee, Laura Schulz, Steve Piantadosi and Roger Levy, for their thoughtful feedback. I am incredibly grateful to my collaborators who contributed to this and related work. They include Ryan Adams, Kevin Ellis, Jon Malmaud, Stephen Muggleton, and Dianhuan Lin. In addition, I would like to thank my colleagues in CoCoSci for their guidance, support, and encouragement in this work. I am grateful to the National Science Foundation Graduate Research Fellowship Program for their funding support. This thesis would not exist if not for the love and support of my parents, Rina and Avi Dechter, and it has been improved greatly by my mother's guidance and insight throughout my time at MIT. Lastly, I would like to thank my wife, Amy, for her support and love; my daughter, Susanna, for being an endless source of inspiration as she learns and develops; and my dog, Ruby, for being a consistent companion in the office and at home.

Contents

List of Figures

List of Tables

Chapter 1

Foreword

Whatever innate knowledge infants possess when they are born (Spelke and Kinzler 2007), they do not have at their disposal anything like the conceptual repertoire of the adult. And the lay adult may stand in a similar — though less extreme — relation to the expert, and the new arrival in a foreign culture to the native. An account of the conceptual change that occurs between infancy and adulthood is one of the central challenges of the cognitive sciences, and such an account remains ellusive.

The aim of this thesis is explore one aspect of this mystery through the lense of computational cognitive science, artificial intelligence and machine learning. We set out with the following question: given our best hypotheses about the natural of mental representations and the goals of learning agents, what is it that allows a learner to *find* the right meaning for a concept — to discover an adequate theory of their world from the infinite space of competing theories — and to do so with the limited time and resources at their disposal.

This goal is motivated by a particular view of conceptual knowledge as consisting of *intuitive theories* expressed in a *language of thought*. On this view, people's knowledge about the world is organized around explanatory theories of particular domains, much like scientific theories, and these theories both contain their knowledge about the domain and guide their further learning and exploration (Carey 1985; Wellman and Gelman 1992; Gopnik, Meltzoff, and Bryant 1997). And the mental representation that supports this knowledge is — like the natural languages in which scientific theories are expressed — a symbolic, relational and compositional formalism.

Computational models of cognition that ascribe to this view — that conceptual knowledge is organized in intuitive theories expressed in an LOT — have in recent years been able to provide a satisfying account of various patterns of reasoning and learning in people. With one notable exception (Ullman, Goodman, and Tenenbaum 2012), these models are at the computational level of analysis[1]. Even for those who believe that much remains to be studied at the computational level, pushing beyond this to the algorithmic level is important for at least three reasons. First, a plausible algorithmic account of learning in an LOT model provides a proof of concept that such a model could be instantiated in the human mind. Second, any algorithmic approach

[1]Throughout this thesis, when we refer to levels of analysis, we are refering to David Marr's "computational," "algorithmic," and "implementation" levels Marr 1982

provides us with tools for studying the predictions and behavior of a computational model, without which it is difficult to compare competing models and hypotheses about them. Finally, algorithmic models have the potential to provide us with insight into the behaviors that we observe when children and adults learn, to help us explain not just what they learn, but also when and in what circumstances.

The tools and techniques I will use in this thesis are largely borrowed from computer science, especially AI. Since its inception, AI has attempted to represent commonsense knowledge in programming languages and to develop algorithms that are able to efficiently reason and learn over those representations. Moreover, the prevalence of intractable combinatorial search tasks in AI problems like planning, constraint satisfaction, and automated game playing has led to the development of general search techniques for these domains Russell and Norvig 2010. A general approach of this thesis will be to attempt to apply these knowledge representations and search techniques within the framework of the *Bayes LOT* hypothesis.

The Bayes LOT hypothesis Expressive formal languages, like their natural language counterparts, provide a natural medium in which to represent various kinds of structured information. Whether the goal is to store and query information as in a database, to formalize communication conventions as in a communication protocol, or to encode precisely our understanding of the laws of nature (for example, in a electronic circuit simulator), computing in complex domains is facilitated by using expressive languages that allow for abstraction, composition, and modularity. The ubiquity of these properities in the formal languages that humans use to represent their knowledge suggests that whatever knowledge representation exists in the human mind must also have these properties as well. This line of thinking has lead to the hypothesis that mental representations are expressed in some sort of *language of thought* (LOT), and that human concepts and theories are best understood as expressions in that LOT.

A separate line of research in computational cognitive science provides mounting evidence that people's learning and reasoning, especially in the face of small amounts of noisy and uncertain information, is well described as probabilistic inference over probabilistic generative models of the world that people hold in their minds. This hypothesis is generally described as a *Bayesian* view of cognition, as it draws parallels between the human mind and the logic of a Bayesian statistician.

A small literature has emerged over the last decade or so that seeks to combine the Bayesian view of cognition and the LOT hypothesis (this literature is discussed in detail in Chapter 4). We term this combination the **Bayes LOT hypothesis**:

> knowledge acquisition is best understood as posterior inference in a probabilistic generative model over expressions in a language of thought.

Overview of this thesis The following two chapters present two very Bayes LOT models through which we have explored algorithmic principles by which the human mind can explore the vast hypothesis space induced by a symbolic and compositional knowledge representation.

The first chapter presents a simple model of concept acquisition as the automated synthesis of functional programs. The primary contribution here is to present the Exploration-Compression algorithm, and through it to study the hypothesis that learning in the space of functional programs can be assisted by learning useful and frequently occuring program fragments. We show that in the multi-task learning setting, our approach succeeds in "bootstrapping" domain-specific language that allows for more efficient exploration of the search space.

The second chapter presents a qualitatively different kind of Bayes LOT model, the Infinite Knowledge Base Model (IKM). The IKM is a probabilistic generative model of first-order deductive knowledge bases with an unbounded number of "latent" concepts. We also present a MAP inference algorithm for learning in the IKM. By framing this inference problem as an abuctive search over proofs of observed data, we are able to use the tools of state-space heuristic search to develop efficient learning algorithms over these knowledge bases.

In the final chapter of this thesis, we apply the IKM to some of the concept learning domains that are commonly studied in the Bayes LOT literature, and discuss how it the IKM relates to previously proposed models. We argue that the IKM is at once simple enough to admit tractable learning algorithms and expressive enough to apply to some of the of complex systems of interrelated concepts that characterize people's commonsense knowledge.

Chapter 2

Bootstrap learning via modular program induction

2.1 Introduction

Many of the problems we want to solve in AI are best cast as search problems. There are many search algorithms, but in order to be practically efficient in the high-dimensional and combinatorial spaces that are characteristic of AI domains, they typically exploit the local or topological structure of the search space. In continuous search spaces, for example, convex optimization and gradient based search methods take advantage of local smoothness to move in promising directions. In combinatorial search spaces, local search algorithms move from one candidate solution to a neighboring one, requiring a suitable notion of neighborhood. Heuristic search techniques take advantage of a search space's topological structure, enabling one to prune off large parts of the solution space guided by an approximate heuristic function.

These search algorithms rely on a human with the expertise to identify the suitable representation, similarity metrics, or search operators that will enable these algorithms to perform successfully. Unfortunately, however, many of the tasks we expect intelligent agents to tackle — from the quotidien (emptying the dishwasher, folding the laundry, navigating city streets) to the intellectual (developing new technologies, proving new theorems) — do not come with such a structured search space. This is particularly true of those problems whose solutions are naturally expressed as computer programs, with their data structures, variables, and control constructs. Given this, this work asks how a suitably structured search space for given domain can be extracted from the space of computer programs.

Naturally, we take as our inspiration the human programmer: how does she, when confronted with the boundless space of programs, ever happen on one that solves the task at hand? There are many ingredients to this ability — the ability to understand the specification of a problem, break it into smaller pieces, create a mental model of a program's operation, etc. — but we focus on one particular ingredient: the programmer's ability to notice repeated useful pieces of code, to package them up as named subroutines, and thus create a function library. Armed with such a library, she manages the complexity of the programs she needs to create. Likewise, the EC algorithm builds a library of reusable program components and places a distribution over these, effectively learning from simple tasks a search space that enables it to solve more complex tasks.

In this work, we try to see how far this idea of reusable function definition can get us. We ask the following questions:

- can an AI system discover the structure latent in the solutions to multiple related problems and thereby bootstrap effective exploration of the search space?

- Can discovering modular program components transform a problem from one in which search is intractable into a problem in which it becomes increasingly feasible with experience?

Given collection of tasks in some domain, the EC algorithm works as follows. It begins with a small library of program primitives and a probability distribution over programs synthesized from this library. On each iteration, it generates programs in order of highest probability according to the library learned in the previous iteration, and it tests whether any of these generated programs solves any of the given tasks. From the solutions it finds, it extracts those functions that most compress these solutions, and it estimates a new distribution over programs with these new functions as primitives in an attempt to match the distribution of empirically observed useful programs.

In using compression as a guide to the choice of representation, we instantiate the idea that good representations are those which minimize the description length of typical elements in the domain. We show that this simple idea allows us to achieve high performance in tasks where just searching according to the initial distribution over program primitives clearly fails. Beyond this, we show how to couch this algorithm within the framework of hierarchical Bayesian inference, demonstrate its application within a variety of disparate domains, present several ways to combine it with more sophisticated search techniques, and provide some theoretical justifications for its efficacy.

The remainder of this chapter is structured as follows: after a discussion of related work in the literature in the next section, we will we elaborate on the general Bayes LOT framework (Section 2.3). In Section 2.4, we present the basic EC algorithm, along with the particular learning setting and generative model for which the EC algorithm is an approximate solution. In Section 2.5, we discuss several variations to the EC algorithm that we have studied. Finally, Section 2.6, we will present results and insights from several case-studies and experiments we have run using these variants of EC.

2.2 Related Work

The idea of discovering and using reusable subcomponents is part of an old tradition in AI. For instance, it is one of the central themes in Herb Simon's 1969 "The Sciences of the Artificial" (Simon 1996). Rendell's 1985 paper (Rendell 1985), "Substantial Constructive Induction Using Layered Information Compression: Tractable Feature Formation in Search," presented within a classical search paradigm the idea that compression provides a heuristic for constructing good representations.

Within the Inductive Logic Programming (ILP) literature, researchers have explored "Constructive Induction," generating new and reusable logic programming terms (Muggleton 1987). Later, predicate invention was also applied in the multitask setting (Khan, Muggleton, and Parson 1998). We apply these ideas to learning functional programs, and believe that the modularity and compositionality afforded by this representation is necessary for these ideas to be successful.

Genetic programming (GP) (Koza 1993) has tackled the problem of program learning and has explored the notion that reusable functions are helpful. This work relies, however, on stochastic local search over the space of programs, and automatically discovering useful subroutines is seen as a helpful heuristic. More recently, Liang et al. (Liang, Jordan, and Klein 2010) used a stochastic grammar over combinatory logic to induce shared structure in multi-task program induction problems. We find this latter representation compelling and use it here, but we see both this and genetic programming as tackling the question of how to benefit from shared structure in situations where local search might be successful on its own. Such problems rely on domain knowledge — whether in the form of a fitness function or a likelihood function — to provide local structure to the search space.

The idea of composing primitive operations in order to construct *macro* operators that allow for more efficient search has been explored extensively in the search, planning, and reinforcement learning literatures. (Fikes, Hart, and Nilsson 1972) introduced the idea of generalizing solutions to STRIPS planning problems; these generalization solutions, created by "lifting" a particular solution by replacing some constants with variables, were called macro operators, or MACROPS. Later, Korf 1985b presented the *Macro Problem Solver*, an approach for solving combinatorial search problems by first automatically synthesizing macro operators and then applying those macro-operators in a deterministic, backtrack-free manner. The use of macros has also been widely explored in the reinforcement learning literature, with some work exploring how macros can be automatically generated (Hauskrecht et al. 1998; Randlov 1999). For the most part, the literature on macro generation in these domains has focused on abstracting macros from plans, solution paths, execution traces, etc. By contrast, EC uses a formalism that explicitly represents abstraction (i.e. higher-order functional programs); at the cost of making the search space much more complex, this choice of representation means that the solution space and the space of "macros" is the same.

A central difference between the work we present here and previous related work is the use of a probabilistic generative model of programs to guide the generation and selection of learned functions. Instead of looking at function learning as merely a technique for generalizing from one solution to another or for learning to make search more efficient, we see function learning as the result of discovering the latent structure of the search space.

2.3 The Bayes LOT framework

In order to motivate our view of multi-task program synthesis as hierarchical Bayesian inference, we elaborate on the Bayes LOT hypothesis, introduced in the Chapter 1. The Bayes LOT hypothesis is the hypothesis that when people learn and reason about the data that they observe in the world, they assume that a) the data generating process is parameterized by an expression in a symbolic and compositional formal language, and b) that this expression is drawn from some probability distribution over expressions in this language. Writing Φ for the parameters of the expression generating process, E for an LOT expression, and D for the observed data, these assumptions are summarized by the equations:

$$E \sim p_{E|\Phi}(\cdot|\Phi) \tag{2.1}$$
$$D \sim p_{D|E}(\cdot|\mathbb{E}). \tag{2.2}$$

The Bayes LOT hypothesis is *not* that such a data generating process actually exists in the world. In some cases, this assumption seems reasonable: when reasoning about other people, it seems reasonable to assume that their actions are the product of knowledge represented in their LOT. In other cases, like when reasoning about the laws of physics, the Bayes LOT hypothesis seems merely to capture the psychological fact that humans behave as if the physical world is determined by such things as laws, that these laws are described in some kind of more or less formal linguistic medium, and that not all laws are a priori equivalently likely to hold.

With this general model in place, the Bayes LOT framework posits that learning, reasoning, prediction, explanation, etc., correspond to the standard applications of Bayes' rule to this probabilistic model.

In order to model how learners can acquire knowledge about the structure of a given domain, we turn this simple generative process into a hierarchical one:

$$\forall \Delta \in \textsc{domains}: \tag{2.3}$$
$$G^{\Delta} \sim p_{G|\Phi}(\cdot|\Phi) \tag{2.4}$$
$$E_i^{\Delta} \sim_{\text{i.i.d}} p_{E|G}(\cdot|G^{\Delta}) \tag{2.5}$$
$$D_i^{\Delta} \sim p_{D|E}(\cdot|E_i^{\Delta}), \tag{2.6}$$

where G^{Δ} captures the abstractions and patterns that are relevant for domain Δ. Since our goal in this work is to examine how this hierarchical setting might support search over latent expressions, we will treat the data that each expression generates as a black box, i.e., we will assume that that there is a known function t where $t(D, E)$ is the "fit" of expression E to data D. This allows us to define *tasks* $t_i(\cdot) = t(D_i, \cdot)$. The generative process is shown as a graphical model in Figure 2-1.

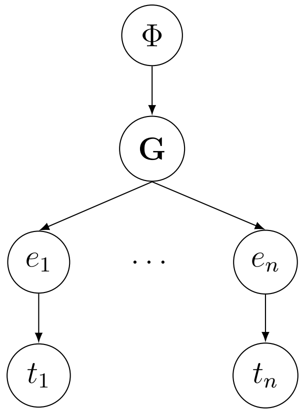


Figure 2-1: Graphical model for the multitask reinforcement learning variant of the Bayes LOT framework. The stochastic grammar G is parameterized by hyper-parameters Φ, and latent expressions $e_1, \ldots, e_n$ are generated i.i.d from G. Each task t_i is parameterized by its associated expression e_i.

2.4 The Exploration-Compression algorithm

In this section, we present the EC algorithm, motivating it as a concrete instantiation of the hierarchical Bayes LOT model presented in the previous section. We first present in more detail the generative model upon which the algorithm is based, and we then describe an approximate scheme for inference in the model. This results in a general version of the EC algorithm. Subsequently, we introduce combinatory logic as the formal language we use, and describe simple implementations of the two fundamental steps – EXPLORATION and COMPRESSION – in the context of this formalism.

2.4.1 Generative Model

Our Bayes LOT learner is presented with a sequence of tasks $\bar{z} = z_1, \ldots, z_n$. Each task is a function $z_i : \mathbb{L} \mapsto \mathbf{R}_{\geq 0}$ that takes an expression in language $\mathbb{L}$ and returns a non-negative reward. The learner assumes that the reward $z_i(e)$ corresponds to some measure of agreement between e and a latent expression e_i. In the simplest case, $z_i^\delta(e|e_i)$ is one if e and e_i correspond to the same function, and zero otherwise; that is $z_i^\delta(e|e_i) = \mathit{1}(\text{func}(e) = \text{func}(e_i))$, where $\text{func}(x)$ is the function associated with expression x[1]. The learner additionally assumes that the latent expressions $e_1, \ldots, e_n$ are drawn independently from a latent stochastic grammar $\mathbf{G}$ over $\mathbb{L}$, and $\mathbf{G}$ is drawn from a distribution over stochastic grammars parameterized by Φ (see Figure 2-1).

[1]This requires some notion of equality of functions, which we will take as defined extensionally, i.e., functions f and g are equivalent if $\forall x \in \Pi . f(x) = g(x)$ over some domain of interest Π

2.4.2 Inference

The learner's task is to jointly estimate the latent expressions $\bar{e} = e_1, \ldots, e_n$. Our algorithm does so by finding a *maximum a posteriori* (MAP) value of $\mathbf{G} = \hat{\mathbf{G}}$, and then returning the MAP value of $e_1, \ldots, e_n = \hat{e_1}, \ldots, \hat{e_n}$ given $\hat{\mathbf{G}}$:

$$\hat{\mathbf{G}} = \arg\max_{\mathbf{G}} \sum_i \log \sum_{\substack{e_i \in \mathbb{L}: \\ z_i(e_i) = 1}} p(e_i | \mathbf{G}) \tag{2.7}$$

$$\hat{e}_i = \arg\max_{\substack{e_i \in \mathbb{L}: \\ t_i(e_i) = 1}} p(e | \hat{\mathbf{G}}) \tag{2.8}$$

The EC algorithm can be seen as a applying an approximate MAP *Expectation-Maximization* (MAP-EM) algorithm to this inference task (Dempster, Laird, and Rubin 1977; Krishnan and McLachlan 1997) where, roughly, the EXPLORATION step of EC corresponds to the expectation step of EM and the COMPRESSION step of EC corresponds to the maximization step of EM.n

In MAP-EM, the objective is to maximize the value of the random variable that influences the data via some extra intermediate latent variables (these latter variables are conventionally referred to as the *hidden data*); in our case, the random variables are the parameters of the stochastic grammar and the hidden data are the expressions for each task. EM is an iterative update algorithm. In our case the iterative step for computing the $i+1$-th estimate of grammar $\mathbf{G}$ given the ith one is:

$$\mathbf{G}_{i+1} \leftarrow \arg\max_{\mathbf{G}} \mathbb{E}\left[\log p(\bar{t}, \bar{e}, \mathbf{G} | \Phi) \mid \mathbf{G}_i\right] \tag{2.9}$$

$$= \arg\max_{\mathbf{G}} \log p(\mathbf{G}|\Phi) + \sum_j \sum_{\substack{e_j \in \mathbb{L}: \\ z_j(e_j) = 1}} p(e_j | \mathbf{G}_i, z_j) \log p(e_j | \mathbf{G}), . \tag{2.10}$$

That is, each $\mathbf{G}_{i+1}$ is the latent grammar that best accounts for observed data (tasks) given the distribution over expressions fixed by the previous grammar estimate $\mathbf{G}_i$.

Of course, we cannot actually perform the summation over all expressions in the language; instead, we enumerate a *frontier* $F(\mathbf{G})$ of the N_f best expressions under the current grammar $\mathbf{G}$, and we modify the sum in Equation 2.10 to be:

$$\mathbf{G}_{i+1} \leftarrow \arg\max_{\mathbf{G}} \log p(\mathbf{G}|\Phi) + \sum_j \sum_{\substack{e_j \in F(\mathbf{G}_i): \\ z_j(e_j) = 1}} P(e_j | \mathbf{G}_i) \log p(e_j | \mathbf{G}) \tag{2.11}$$

Solving for $\mathbf{G}_{i+1}$ in Equation 2.11 involves two basic steps:

1. EXPLORATION: We need to generate the frontier $F(\mathbf{G}_i)$, and, for each task, col-

lect those expressions in the frontier that solve it; we call this the EXPLORATION step.

2. COMPRESSION: We need to find the grammar parameters that maximize the log probability of this grammar plus the log probability of task-solving expressions that we have collected, given these grammar parameters. It is natural in the context of distributions over expressions to interpret the log probability of an expressions as the negative description length (roughly, the number of bits). Essentially, then, this second step is to find the grammar $\mathbf{G}$ that minimizes the sum of its description length plus the description length of the task-solving expressions in the frontier, when encoded by the grammar. In this way, we operationalize the maximization in Equation 2.11 as a compression operation, and we call this operation the COMPRESSION step of the algorithm.

Let $F_j(\mathbf{G})$ be the set of expressions in the frontier that satisfy task z_j: $F_j(\mathbf{G}) \triangleq \{e | e \in F(\mathbf{G}), z_j(e) = 1\}$. Then, the EC algorithm can now be stated as:

Algorithm 1: The EC algorithm.

```
for $i \in 1, 2, \ldots$ do
    $F^{(i)} \leftarrow \mathrm{nBest}(N_f, G_i)$ ▷ Exploration
    $\mathbf{G}_{i+1} \leftarrow \mathrm{compress}(\mathbf{G}_i, F_1^{(i)}, \ldots, F_n^{(i)})$▷ Compression
end
```

2.4.3 Language

Our implementation of the Algorithm 1 requires us to specify a representation language for expressions and a stochastic grammar over expressions in this language. Here, we choose to use combinatory logic (Schönfinkel 1967; Curry and Feys 1968). Like the lambda calculus, it is a kind of minimal functional language. Specifically, we use a polymorphic simply-typed combinatory calculus, which can be seen as a variable-free subset of the polymorphic simply-typed lambda calculus (Pierce 2002).

A well formed expression in the combinatory logic, known as a *combinatory term*, is either a *combinator* P or the application $(E_1 E_2)$ of one combinatory term to another[2]. The set of combinators in a specific calculus is its *basis*. Each combinator P is associated with a rewrite rule of the form $(P x_1 \ldots x_n) \rightarrow E$ where the x_i are variables that stand for combinatory expressions, and E is a combinatory expression containing these variables. Thus, the combinators – and by extension, any combinatory term – is a function on combinatory terms. Some common combinators are defined as follows:

$$\mathbf{I}\, x \rightarrow x \qquad \text{(identity)} \tag{2.12}$$

$$\mathbf{S}\, f\, g\, x \rightarrow (f\, x)\, (g\, x) \tag{2.13}$$

$$\mathbf{C}\, f\, g\, x \rightarrow (f\, x)\, g \tag{2.14}$$

$$\mathbf{K}\, f\, g \rightarrow f \tag{2.15}$$

$$\mathbf{B}\, f\, g\, x \rightarrow f\, (g\, x) \qquad \text{(composition)} \tag{2.16}$$

[2]When parentheses are omitted, application is assumed to associate to the left, e.g. $abc = ((a\, b)\, c)$.

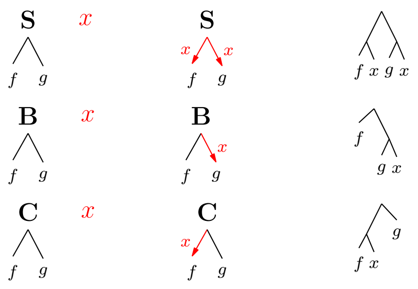


Figure 2-2: Combinators implement variable-free language via the *routing* behavior of **S**, **B**, and **C**.

A basis is *complete* if it allows for the expression of any Turing-computable function. Although the basis combinators shown above constitute a complete basis, we will allow our language to have greater variety of primitive combinators, capturing various kinds of background knowledge. Note that by default we consider all primitives to be in their "curried" form (e.g. a function like "+" adds two numbers x and y by being first applied to x, returning the function $(+\,x)$ and then being applied to y, returning $((+\,x)\,y)$).

The basis combinators can themselves be expressed in the lambda calculus (with definitions following directly from the equations above). The lambda calculus has two basic operations – application and abstraction – but in using the combinatory logic we sequester uses of the abstraction operation inside the combinators. In doing so, we have replaced the variable binding of the λ operator with the variable *routing* of these basis combinators; this routing behavior is demonstrated in Figure 2-2. Our representation thus becomes variable-free. See Liang, Jordan, and Klein 2010 for a more detailed discussion of this routing interpretation.

Using combinatory logic is very convenient for program synthesis (Briggs and O'Neill 2008): since every expression is the application of one expression to another – with this recursion bottoming out at the primitive combinators – so each program is a binary tree. Most importantly, any subtree is itself a well-formed expression; this is not the case in the lambda calculus, since abstraction introduces long range dependencies between the λ operator and the variables to which that operator refers. In the lambda calculus, then, a subtree might have free variables, not bound to any enclosing λ.

As a simple example of our representation, consider the squaring function, represented in the lambda calculus, $\lambda x. * x\, x$. Using two of the basis combinators above, we can write the squaring function in combinatory logic as $\mathbf{S} * \mathbf{I}$ (taking $*$ as a background combinator). When we apply this combinator to a value x, the action of the combinator is defined as $\mathbf{S} * \mathbf{I}\, x \;\rightarrow\; (*\,x)\,(\mathbf{I}\,x) \;\rightarrow\; *\,x\,x$.

We can extend this representation with a simple polymorphic type system (Pierce 2002). In this representation, a type t is either a type primitive (e.g. reals, integers, booleans, etc.), a type variable (e.g. σ), or a function type $t_1 \rightarrow t_2$ of functions from source type t_1 to target type t_2. Any term can be represented as a binary tree (see Figure 2-3) whose leaves are typed primitive combinators and whose interior nodes

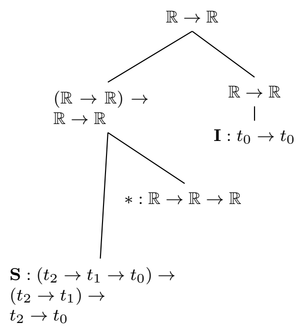

$\mathbb{R} \to \mathbb{R}$
$(\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R}$
$\mathbb{R} \to \mathbb{R}$
$\mathbf{I} : t_0 \to t_0$
$* : \mathbb{R} \to \mathbb{R} \to \mathbb{R}$
$\mathbf{S} : (t_2 \to t_1 \to t_0) \to (t_2 \to t_1) \to t_2 \to t_0$

Figure 2-3: The typed combinator $\mathbf{S} * \mathbf{I}$, which computes $f(x) = x^2$ over real values $\mathbb{R}$, represented as a binary tree. Nodes are annotated with their types, i.e. if node n has left child $\ell : \tau$ and right child $r : \sigma$ then the type of n is $\mathrm{mgu}(\tau, \sigma \to \pi)$.

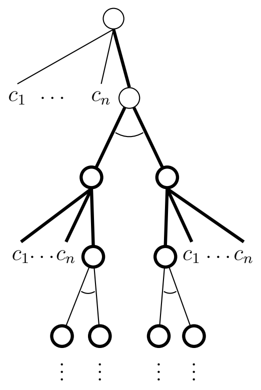


Figure 2-4: The space of all typed combinators represented as an infinitely deep AND/OR tree.

represent typed applications of one term to another. We write $e : t$ to mean that expression e has type t.

2.4.4 A stochastic grammar over programs

To define the stochastic generative process over expressions given $\mathbf{G}$, we introduce some basic concepts and notation.

A *fresh type variable* in a scope is a type variable that does not occur in any type expression in that scope. A *type substitution* Σ is a partial function from type variables to type expressions, and when applied to a type expression t, which we write as $t[\Sigma]$ it replaces each occurance in t of a type variables σ in the domain of Σ with $\Sigma(\sigma)$. We write a substition as a set of bindings from type variables to type expressions, e.g., $\{\}$ is the identity substitution, $\{\sigma \mapsto t\}$ is a substition with a domain consisting of the single type variable σ. A *unifier* of two type expressions t and s is a substitution Σ such that $t[\Sigma] = s[\Sigma]$; if such a unifier exists, we say that t and s *unify.* Given two type expressions t and s that unify, the function $\textsc{mgu}(t, s)$ returns the *most general unifier (mgu)* of t and s. A unifier Σ of t and s is the mgu of t and s if for any unifier Ω there is a unifier Δ such that $\Omega = \Sigma \circ \Delta$.

A *typed combinator* is a combinator associated with a type expression. The type of an combinatory term is defined by a partial function type$(\cdot)$: if e is a combinator, then type(e) is just its associated type. If e is an application of the form $(e_\ell\, e_r)$, then type$(e) = t\Sigma$ where $\Sigma = \textsc{mgu}(t_r \mapsto t, t_\ell)$ if such a unifier exists; otherwise, type(e) is undefined. If type(e) exists then we say that e is *well-typed*.

We define a *stochastic combinatory grammar* $\mathbf{G} = (\bar{c}, \bar{w}, w_{\text{app}})$ to be a stochastic grammar (Booth and Thompson 1973) where *library* $\bar{c}$ is a collection of M typed combinators, $\bar{w}$ is a collection of M *library weights*, and w_{app} is the *application weight*:

$$\bar{c} = \{c_1, \ldots, c_M\}, \tag{2.17}$$
$$\bar{w} = \{w_1, \ldots, w_M\}, \text{ where } w_i \geq 0 \tag{2.18}$$
$$w_{\text{app}} \in [0, 1]. \tag{2.19}$$

A *derivation* is a binary tree whose leaves are combinators and whose nodes are either *application nodes* or *combinator nodes*. Combinator nodes may only appear at the leaves of the derivation. In a *complete* derivation, all leaf nodes are combinator nodes; if a derivation is not complete we call it a *partial* derivation. Application nodes are labeled with type expressions, while combinator nodes are labeled with combinators.

The *yield* of an application node n, written $\textsc{yield}(n)$ is the application of the yield of n's left child to its right child. The yield of a combinator node is simply the associated combinator. The yield of a derivation is the yield of its root node. We define the type of a derivation to be the type of its yield, i.e., for derivation d, type(d) = type$(\textsc{yield}(d))$. For reasons that will be clear, the label of a node in a derivation is called its *requesting type*, which we write $\textsc{ReqTy}(n)$. We write an application node with requesting type t, left child ℓ and right child r as $\textsc{node}(t, \ell, r)$; we write a terminal node with requesting type t and combinator c as $\textsc{node}(t, c)$.

Algorithm 2 shows the generative process for sampling derivations given grammar $\mathbf{G}$ given requesting type t. That is, Algorithm 2 defines a distribution over derivations terms conditioned on their types unifying with a particular requesting type. Under this process, derivations are generated top-down and left-to-right. The function $\textsc{Gen}(t)$ generates a derivation whose root node has requesting type t. With probability w_{app} it samples a left child derivation whose requesting type is the type of functions from some type variable σ to the parent requesting type t. Once this left child is sampled, any resulting variable bindings induced by the left child are applied to σ and the right child of the derivation is sampled conditioned on being the proper input type for the left child. With probability $1 - w_{\text{app}}$, the recursive descent is terminated by a combinator node whose combinator is sampled from those combinators in the library whose types unifies with the requesting type t, in proportion to their weights.

Algorithm 2: The generative process for sampling a combinatory term of requesting type t from stochastic grammar $\mathbf{G} = (\bar{c}, \bar{w}, w_{\text{app}})$.

```
$\Sigma \leftarrow \{\}$
Def Gen($t$):
    $x \sim \text{Bernoulli}(w_{\text{app}})$
    if $x = 1$ then                                  /* sample application node */
        $n_\ell \leftarrow$ Gen($\sigma \mapsto t$)  where $\sigma$ is a fresh type variable
        $n_r \leftarrow$ Gen($\sigma[\Sigma]$)
        return node($t, n_\ell, n_r$)
    else                                              /* sample combinator node */
        sample $i \propto w_i \mathbf{1}[t \text{ unifies with } t_i]$        /* fails if no comb. with
        unifying type */
        $\Sigma \leftarrow \Sigma \circ$ mgu($t, t_i$)     /* extend substitution with most general
        unifier */
        return node($t, c_i$)
    end
return Gen($t$)
```

2.4.5 The EXPLORATION step: best-first enumeration of programs

Algorithm 1 requires a procedure nBest which returns the frontier $F(\mathbf{G})$ — the N_f expressions with the highest prior probability under the grammar $\mathbf{G}$. There has been recent interest in this problem, most of it focused on enumerating the shortest program satisfying some criterion (Katayama 2005; Yakushev and Jeuring 2009). Our enumeration procedure is best described by the following recursion: every program is either a primitive combinator or it is a left child program applied to a right child program.

We formulate this procedure as a best-first exploration of *derivation state space*, the state-space graph of partial derivations whose requesting type is t. $\mathcal{D}(t)$ is defined as follows:

1. the nodes of $\mathcal{D}(t)$ are partial derivations;

2. the root of $\mathcal{D}(t)$ is labeled by a single derivation node with requesting type t;

3. the solution nodes of $\mathcal{D}$ are those that are labeled by complete derivations;

4. the child of a node a labeled with a partial derivation d is obtained as follows: let n with requesting type s be the left most non-terminal of d; then child b of a is a obtained by connecting n to either a) a left and right nonterminal of types $\sigma \mapsto s$ and σ, respectively, or b) a terminal node labeled with a combinator that unifies with s. In the latter case, the unifier is applied to the remaining non-terminal nodes in the derivation.

For each node in $\mathcal{D}(t)$ labeled by partial derivation d, we define its cost to be its negative log probability. This allows for a straight-forward application of breadth-first search for enumeration of the frontier.

2.4.6 The COMPRESSION step

Once the EXPLORATION step has enumerated a frontier of expressions, the COMPRESSION step of the EC algorithm attempts to find a new stochastic grammar such that description length of the grammar plus the expressions that solve the given tasks is minimal.

One way to do this is to describe a distribution over stochastic grammars and perform posterior Bayesian inference over that distribution. Such an approach, using an *adaptor grammar* — a non-parametric Bayesian distribution over PCFGs — is described in Appendix A. Though theoretically elegant, the approach we will present in this section is a heuristic approximation.

From Equation 2.10, the COMPRESSION step of EC is finding the grammar that solves the following optimization problem:

$$\arg\max_{\mathbf{G}} \log p(\mathbf{G}|\Phi) + \sum_j \sum_{e_j \in F_j(\mathbf{G}_i)} p(e_j|\mathbf{G}_i) \log p(e_j|\mathbf{G})$$

We can rewrite this optimization problem as:

$$\arg\min_{\mathbf{G}} -\log p(\mathbf{G}|\Phi) + \sum_{e \in F(\mathbf{G}_i)} -\log p(e|\mathbf{G}) \times \sum_j q_{i,j}(e) \tag{2.20}$$

$$\text{where} \quad q_{i,j}(e) = p(z_j|e)p(e|\mathbf{G}_i)/Z_{i,j} \tag{2.21}$$

$$Z_{i,j} = \sum_{e \in F(\mathbf{G}_i)} p(z_j|e)p(e|\mathbf{G}_i) \tag{2.22}$$

Here one should interpret $q_{i,j}(e)$ as the responsibility of program e for explaining task j on iteration i of the EC algorithm.[3] We have defined $Z_{i,j}$ to ensure that $q_{i,j}(e)$ is normalized.

Equation 2.20 has a natural interpretation as a form of compression. It says to jointly minimize the description length of the grammar (i.e., $-\log p(\mathbf{G}|\Phi)$) plus the description length of every program in the frontier (i.e., $-\log p(e|\mathbf{G})$) where the description length of a program is scaled by the expected number of times that it was used to solve a task (i.e., $\sum_j q_{i,j}(e)$). At this point we can use any grammar induction algorithm to solve Equation 2.20.

Inducing a grammar breaks down into two steps: (1) inferring the symbolic structure of the grammar, which for EC means deciding which combinators are cached; and then (2) estimating the real-valued parameters of the grammar, which for EC means estimating the terminal weights $\bar{w}$ and application weight w_{app}.

[3]The first presentation of EC (Dechter et al. 2013) used a slightly different but related approach: the approximating distributions $q_{i,j}(e)$ were bootstrapped without $\mathbf{G}_i$ by searching for an assignment of programs to tasks that maximize reuse of subexpressions. See Dechter et al. 2013 for details.

Grammar Structure Learning

An especially simple and tractable algorithm for inducing the structure of a grammar minimizing Equation 2.20 is the Nevill-Manning algorithm (Nevill-Manning and Witten 1997). Nevill-Manning incorporates into the grammar any subtree that is used more than once. But for our grammar induction problem, the observed programs e are also annotated with a real-valued weight telling us how many times we expect to see e as a solution to a task ($\sum_j q_{i,j}(e)$). A natural generalization of Nevill-Manning to our setting is to incorporate into the grammar any subtree whose expected number of occurrences is at least some constant λ:

$$e \in \mathbf{G} \text{ whenever } \lambda \leq \sum_{e' \in F(\mathbf{G}_i)} k(e,e') \times \sum_j q_{i,j}(e') \tag{2.23}$$

$$\text{where} \quad k(e,e') = \# \text{ times } e' \text{ occurs in } e \tag{2.24}$$

The constant λ should be interpreted constant that controls how much we regularize the structure of the grammar. Increasing λ favors smaller grammars; decreasing λ favors putting subtrees into the grammar that were used in programs which solved the tasks. The original Nevill-Manning algorithm corresponds to $\lambda = 1$.

An alternative derivation of this modified Nevill-Manning algorithm arises from the following approximations to the description lengths of $\mathbf{G}$ and $e|\mathbf{G}$:

$$-\log p(\mathbf{G}|\Phi) \approx \lambda \sum_s 1[s \in \mathbf{G}] \tag{2.25}$$

$$-\log p(e|\mathbf{G}) \approx \sum_s 1[s \notin \mathbf{G}] \times k(s,e) \tag{2.26}$$

where s ranges over all possible subtrees of all programs in the frontier. Using this approximation the total description length can be rewritten as

$$-\log p(\mathbf{G}|\Phi) + \sum_{e \in F(\mathbf{G}_i)} -\log p(e|\mathbf{G}) \times \sum_j q_{i,j}(e) \tag{2.27}$$

$$\approx \sum_s 1[s \notin \mathbf{G}] \left(-\lambda + \sum_e \sum_j q_{i,j}(e) k(s,e) \right) + \text{const.} \tag{2.28}$$

which has as its unique minimum the same solution has Equation 2.23.

Grammar Parameter Estimation

With the discrete structure of the grammar in hand, we need to estimate the continuous parameters of the grammar. For EC this is the probability associated with each cached subtree along with the probability of application, w_{app}. Because our grammar is not context-free, this parameter optimization problem is not tractable: the probability that a combinator is used as a terminal at some node depends on the set of other combinators that unify with the requesting type at that node. This set of competing combinators are different depending on the requesting type, resulting in a highly non-convex optimization problem. In the Appendix (Section A), we discuss an intricate but principled approach to resolving this problem.

In practice, however, we can use parameter estimation algorithms designed for stochastic context free grammars in order to estimate the parameters of our context-sensitive grammars. We adopted this scheme and regard it as an approximate yet effective heuristic. Our approach is an EM algorithm that works by alternating between estimating a distribution over derivations (parses), and estimating the parameters of the grammar conditioned on the parses.

Given a grammar structure $\mathbf{G}$ along with its parameters $\bar{w}$, EM assigns the following distribution over parse z of expression e:

$$q(z|e, \mathbf{G}, \bar{w}) \propto 1[z \text{ is a derivation of } e] p(z|\mathbf{G}, \bar{w}) \tag{2.29}$$

Given these distributions over parses, EM updates the weight w_e of an expression $e \in \mathbf{G}$ having type τ to be:

$$w_e \leftarrow \frac{\mathbb{E}_{q_{i,j}, q_{z|e,\mathbf{G},\bar{w}}}[\#\text{ times } e \text{ occurs in } z]}{\mathbb{E}_{q_{i,j}, q_{z|e,\mathbf{G},\bar{w}}}[\#\text{ times requested type unifies with } \tau]} \tag{2.30}$$

Intuitively, Equation 2.30 says that we estimate the probability of using expression e by dividing the number of times that it was used by the number of times that it could have been used. Efficiently computing the expectations in Equation 2.30 over the $q(z|e, \mathbf{G}, \bar{w})$ distribution given in Equation 2.29 is a well studied problem in the Natural Language Processing literature, where a common solution is to use the *Inside/Outside algorithm* (Manning, Schütze, et al. 1999). We adopt this approach. Inside/Outside is a dynamic program that, in our case, recurses on subexpressions of programs in the frontier. We refer the reader to Manning, Schütze, et al. 1999 for more detail.

2.5 Variations on the EC algorithm

In this section we describe a few ways of augmenting the EC algorithm (1) to handle large frontiers (Section 2.5.1); (2) to build programs in an incremental fashion (Section 2.5.2); and to induce programs that humans can interpret (Section 2.5.3). Readers can safely skip ahead to Section 2.6 for the experimental results, some of which use these EC variants.

2.5.1 Enumerating the frontier

Memory-efficient frontier enumeration

The Explore step of the EC algorithm must enumerate a very large frontier of candidate programs. Best-first enumeration has memory requirements that scale linearly with the size of the frontier. In practice memory rather than time can become the bottleneck. Two alternative enumeration schemes sidestep this problem:

Iterative deepening. One may maintain the same asymptotic time complexity of best-first enumeration using iterative deepening. Rather than increasing the search radius by one discrete choice, we increase the maximum *description length* of the enumeration by a constant amount of entropy. For a given description length bound, we can use depth first enumeration to consider every program whose description length under the current generative model is appropriately bounded.

Monte Carl0 EM. The compression step is to find the grammar $\mathbf{G}^*$ as follows:

$$\mathbf{G}^* = \arg\max_{\mathbf{G}'} \mathbb{E}_{\bar{e}\sim\mathbf{G}} \left[\log\left(p(\bar{t}|\bar{e})p(\bar{e}|\mathbf{G}')\right)\right] + \log p(\mathbf{G}'|\Phi)$$

We can approximate the expectation over $\bar{e} \sim \mathbf{G}$ through likelihood weighting, yielding a Monte Carlo EM algorithm where the missing data ($\bar{e}$) is sampled from the previous grammar ($\mathbf{G}$) — replacing enumeration with sampling. This stochastic scheme fails to encourage reuse of subexpressions as strongly as deterministic enumeration.

2.5.2 Sequentially Constructing Programs by Sequential EC

EC's generate-and-test search algorithm works by proposing whole programs and evaluating them against the task set. However, when our tasks come equipped with likelihood functions that give partial credit, we might more intelligently explore the space of programs by *sequentially* constructing programs by repeatedly modifying solutions that get lots of partial credit on a task. Here we describe a way of adapting the EC algorithm to this setting, giving the Sequential EC algorithm.

Sequential EC works by first enumerating a frontier exactly like EC. But then, it heuristically searches through the space of programs, searching for programs that achieve high likelihood for a given task. Each program in the search space is either a program that was originally in the frontier, or a composition of programs that were originally in the frontier. So the transitions we can take in the search space are given by function composition. See Algorithm 3, which explores the search space with beam search. After searching for solutions to all of the tasks we estimate a new grammar exactly as the unmodified EC algorithm does.

Algorithm 3: The search algorithm used to enumerate a frontier in the Sequential EC algorithm. Frontier enumeration is guided by the likelihood model of each task, so we run this search algorithm independently for each task. This is a beam search over programs where the beam width has size N_f and the search has depth L.

```
SequentialECFrontierEnumeration($\mathbf{G}$, frontier size $N_f$, task $t$ with type $\tau$, search bound $L$
$F_1$ = nBest $(N_f, \mathbf{G}, \tau)$ ▷ First frontier comes from grammar $\mathbf{G}$, programs have type $\tau$
$T$ = nBest $(N_f, \mathbf{G}, \tau \to \tau)$ ▷ Transitions in search space of type $\tau \to \tau$
for $i = 2, 3, 4, \ldots, L$ do
    $H_i = \{f(x) | x \in F_{i-1}, f \in T\}$
    $F_i$=top $N_f$ members of $H_i$ as measured by joint probability $p(t|\cdot)p(\cdot|\mathbf{G})$
end
return $\bigcup_{i=1}^{L} F_i$
```

2.5.3 Symbolic Dimensionality Reduction

One of the major advantages of using a programming language as a knowledge representation is that symbolic languages are often readily interpretable by people. This suggests the knowledge learned by EC might be useful not just for searching for programs that solve various tasks, but also for helping people understand the structure of a given domain. This application of EC – which we call *symbolic dimensionality reduction* — attempts to extract a compressive symbolic representation from a dataset, ideally one which is interpretable by humans. We adopt LOT expressions as the symbolic representation.

In the setup we have presented so far, however, the output of EC would be the learned grammar, and for each task (i.e. datum), an expression. However, it is often more natural to think of data as the product of a single program applied to various different inputs. To use EC in this application, then, we modify our generative model to the one shown in Figure 2-5; the latent expressions are factored into task-specific argument expressions (the a_i) and a domain-specific program d, such (da_i) evaluates to c_i. For symbolic dimensionality reduction, we take as our goal to infer the latent program d, marginalizing over the corresponding arguments a_i, conditioned only on the observed tasks z_i.

For an observed dataset consisting of the tasks $\{z_i\}_{i=1}^N$, the best program d^* is given by

$$d^* = \arg\max_d \left(\log P(d|\mathbf{G}_0) + \sum_{i=1}^{N} \log \sum_{\substack{a: \\ d(a)=z_i}} P(a|\mathbf{G}_0) \right) \tag{2.31}$$

where $\mathbf{G}_0$ is a base grammar. This objective function finds the d^* that maximally compresses $\{z_i\}_{i=1}^N$, which corresponds to MAP inference in the model diagrammed in Figure 2-5.

Exact computation of the sum over a in Equation 2.31 requires summing over the

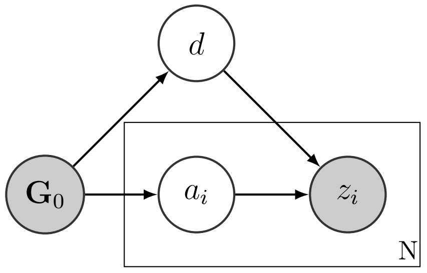


Figure 2-5: Generative model for symbolic dimensionality reduction

infinite space of all programs. This is impossible, so we instead use EC to find sets of programs that solve each task, $\{F_i\}_{i=1}^N$, that are likely to share common structure. This pre-training permits tractable maximization in Equation 2.31: for each program in $\bigcup_{i=1}^N F_i$, there is at most one candidate value of d^* which must be checked against at most $|\bigcup_{i=1}^N F_i|$ other programs. By keeping a constant number of most likely programs under the grammar, we have that $|F_i| = O(1)$ and that this approximate inference of d^* takes time quadratic in N.

2.6 Applications

In this section, we present case studies in which we explore the algorithmic variations discussed above to various tasks and domains. Doing so allows us to experimentally study the algorithm's behavior, examine its practical applicability, and explore ways in which it could be extended or improved.

Note that, for the sake of interpretability, we will often display programs and program fragments as lambda-calculus expressions even though all experiments below used the combinatory calculus as the representation formalism. The translation of an expression from combinatory calculus to lambda-calculus is straightforward and can be done automatically, but the lambda-calculus is far more human-readable than combinatory calculus.

2.6.1 Applications of Vanilla EC

We begin with two studies of Vanilla EC on two domains designed to reveal to what extent the various pieces of the algorithm contribute to its learning capabilities.

Symbolic regression

In this first empirical case-study, we explore the performance of the vanilla EC on a domain consisting of symbolic regression problems. For each task in this problem, we want to find an algebraic function, symbolically specified, that maps a set of input values to output values. We choose this problem because it has a long history in AI, particularly in the genetic programming literature (Koza 1993).

In the formulation we consider, each task t corresponds to a polynomial f; an expression e solves t if it returns the same value when applied $i \in 0, \ldots, 9$ that f does. In the experiment shown here, the set of tasks t corresponds to the set of all polynomials with degree less than or equal to two and integer coefficients between 0 and 10:

$$t = \{ax^2 + bx + c | a, b, c \in 0, \ldots, 9\}. \tag{2.32}$$

We initialize the learner with a base grammar consisting of the basic combinators I, S, B, C and four arithmetic primitives, 1, 0, $*$, and $+$. Note that the frontiers generated by the EXPLORATION step depend on the initial production weights in the grammar. To get a general sense of the algorithm's performance, we set the initial weights to be slightly different on each run, perturbing them around a uniform weighting: for c_i in the base grammar, we set $w(c_i) \propto 1 + \mathcal{U}(0, 0.1)$.

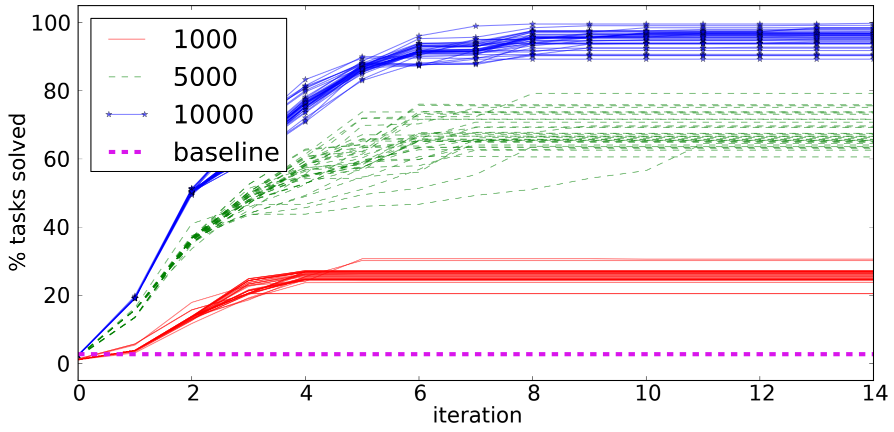


Figure 2-6: Learning curves as a function of frontier size. As frontier size is increased, curves plateau closer to 100% performance. A baseline search over 150000 expressions only hits 3% of the tasks.

Figure 2-6 shows performance results as we vary the frontier size N_f. Changing the frontier size changes the number of tasks the algorithm identifies correctly over the course of 15 algorithm iterations.

A key question is the extent to which EC's grammar re-estimation procedure actually improves the ability of the EXPLORATION step to effectively explore the space of programs. To evaluate this, we can simply look at the difference between the efficacy of a single iteration of EC and multiple iterations. Of course, we need to equate the total number of programs explored by both versions. Thus, as a baseline, we enumerated $10000 * 15 = 150000$ expressions from the initial grammar. This is the total number of expressions that a run of the algorithm sees if it has a frontier size of 10000 and runs for 15 iterations. Thus, if the benefit accruing to the runs with larger frontier sizes is due to simply an increase in the number of expressions seen, rather than increased learning, we should see similarly good performance from this baseline

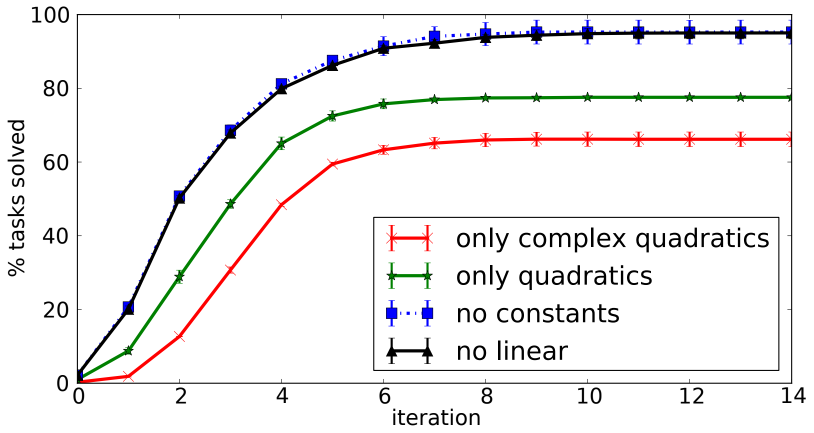


Figure 2-7: How do different task sets affect learning curves? Learning curves at frontier size of 10000 for different task sets.

run. Figure 2-6 clearly shows that this is not the case. In fact, the baseline version only hits 27 of the 1000 tasks (3%), whereas our algorithm nears 100% for a frontier of 10000.

What does the EC algorithm learn in this task? Inspecting the top weighted primitives of the final grammars, we find many incrementers (e.g. (+1), (+2), etc.), several versions of expressions that translate to functions like $x * f(x)$ and $f(x * f(x))$, and combinations of these, like $x * (x + 3) + 3$ and $x * (x + 2)$. Informally, we see expressions that apply functions to themselves, building up complex functions with relatively few unique primitives.

To what degree is "conceptual bootstrapping" responsible for the improved performance? That is, to what extent does the presence of simple problems account for the ability to learn complex functions? One hypothesis is that to succeed on a set of tasks, the set must contain a "learnable" curriculum, a set of tasks that serve as stepping stones from an impoverished representation to a rich one. We can test this hypothesis by varying the set of tasks to which the EC algorithm is exposed. If it is true, then we should see a nonlinear response to reducing the number of easy tasks, as the curriculum changes from being learnable to being unlearnable.

In Figure 2-7, we present learning curves (frontier size 10000) corresponding to various versions of our original symbolic regression task set. Recall that the original set consisted of all polynomials of degree two or less and with coefficients between 0 and 9. In the "no constant" task set, we remove the 9 constant functions in this set. In the "no linear" task set, we remove 90 linear functions from the set. We observe that performance on those task sets does not decrease. However, when we remove both the linear functions and the constant function ("only quadratics"), we see a sharp drop in the algorithm's performance. When we further restrict the task set to only "complex" quadratics (which we define as quadratics whose coefficients are greater than zero), we observe another comparable drop in performance. When we go one step further and restrict the task set to quadratics with coefficients greater than 1, performance drops to 0 because no task is hit in the initial frontier.

This data has in it the nonlinearity that we predicted — that a minimal curriculum of simple tasks is sufficient to achieve high performance — but also suggests that, at least in this domain, this is not an all-or-none effect. Though the performance drops significantly once no linear functions are included in the task set, learning does still occur, and the learning curves still have the same basic shape.

Boolean function learning

In our second case-study of the vanilla EC algorithm, we investigate the algorithm's ability to learn Boolean functions. It is well known that a Boolean circuit can be constructed for any boolean function given only the NAND gate. Our question here is whether the EC algorithm can discover more useful components out of which to build a set of Boolean circuits. Therefore, in this case-study the base grammar consists of the combinators I, S, B, C and the NAND function.

To evaluate how EC's performance is affected by the distribution of problems in the domain, we constructed two task sets, Z_{sampled} and Z_{all}. Z_{sampled} was constructed explicitly to contain familiar modular structure; by contrast Z_{all} simply contains all Boolean functions up to a fixed size.

Experiment 1: learning from sampled Boolean circuits To construct Z_{sampled}, we sampled 1000 Boolean circuits using AND, OR, and NOT gates. To accomplish this, we first sampled either 1, 2, 3 inputs, then between 1 and 5 gates, randomly wiring the inputs of each new gate to the output of one of the existing gates in the circuit. We continued this sampling procedure until we had 1000 "connected" circuits, i.e., circuits all of whose outputs are wired to an input except for the last one (the output gate). This set of one thousand circuits, consisted of 82 unique Boolean functions (i.e. unique truth tables). The distribution of these tasks is visualized in Figure 2-9 (a). For more details on this distribution, see Appendix A.1.

Z_{sampled} consists of the set of tasks that correspond to learning the functions associated with these sampled circuits. That is, given the set C of the 1000 sampled circuits, Z_{sampled} is defined as:

$$Z_{\text{sampled}} = \{t(\cdot|c) | c \in C\} \tag{2.33}$$

$$\text{where } t(e|c) = \mathbf{1}(\text{truth\_table}(e) = \text{truth\_table}(c)). \tag{2.34}$$

In Figure 2-8, we show the learning performance of EC on Z_{sampled} for three frontier sizes, 100, 500, and 1000. The procedure for generating this data is that same as that described in Section 2.6.1. There is a large jump between the performance at frontier size 100 and 500 and a much smaller one between 500 and 1000. Regardless of the frontier size, it we see that there is clearly a suboptimal learning trajectory leading to two distinct learning plateaus. This suggests that it may be difficult for EC to move between local modes.

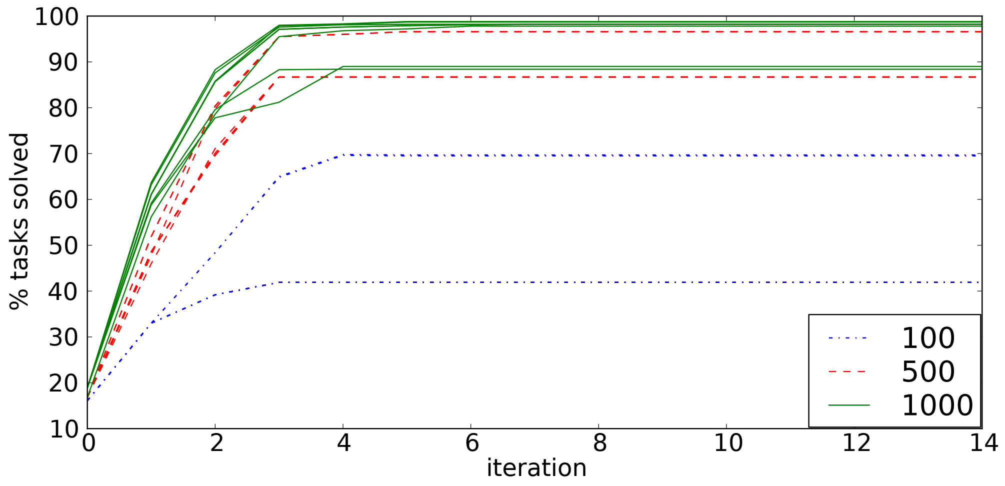


Figure 2-8: Learning curves from Boolean function learning Experiment 1. Learning curves for several frontier sizes on Z_{sampled}, a task set of Boolean functions generated by sampled Boolean circuits.

We can also look at how the learner's implicit distribution over Boolean functions changes over the course of learning. We do so by examining the distribution of Boolean functions that occur in the learner's frontier before and after learning and comparing this to the ground truth distribution. As mentioned above, Figure 2-9 (a) shows the ground truth distribution, while Figures 2-9 (b) and (c) show the distributions before and after learning, respectively (this is on an arbitrarily selected learning run with frontier size 1000). This comparison demonstrates that the distribution over Boolean functions enumerated from the grammar over expressions is much more similar to the true function distribution after learning than before.

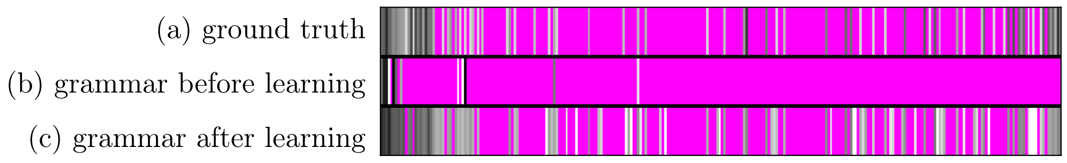


Figure 2-9: Comparing the frontier before and after learning on distribution over boolean circuits. Each row represents a distribution over Boolean functions $\{f | f : \{0,1\}^d \mapsto \{0,1\}, d \in 1,2,3\}$. There are 276 such functions and they are arrayed left-to-right, with $d = 1$ on the left and $d = 3$ on the right. Vertical bands indicate the fraction of expressions/circuits that are equivalent to the corresponding function; a darker band indicates a larger proportion. For clarity, where the proportion is zero, the color is pink.

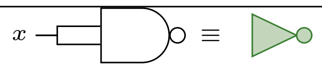
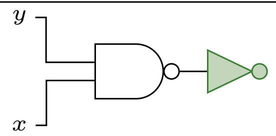
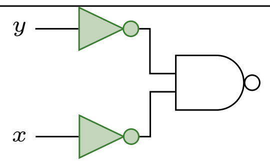
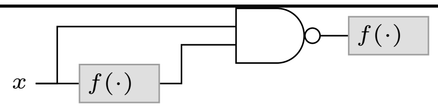
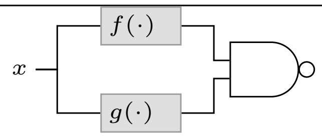
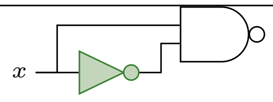
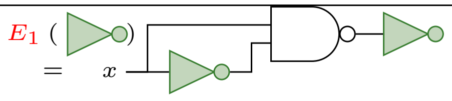
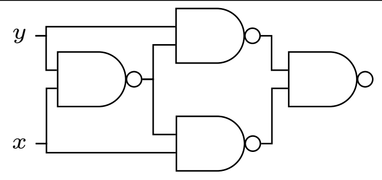
	func.	CL expression	schematic
	NOT	(S NAND I)	
(a)	AND	(C B NAND) (B (S NAND I)) $\rightarrow$ (C B NAND) (B NOT)	
	OR	((B (C (B (S NAND) NAND))) (S NAND I) $\rightarrow$ ((B (C (B (S NAND) NAND))) NOT	
(b)	E_1	S B (S NAND)	
	E_2	(B (C B NAND) S)	
	TRUE	(S NAND) (S NAND I) $\rightarrow$ (S NAND) NOT	
(c)	FALSE	(S B (S NAND)) (S NAND I) $\rightarrow$ E_1 NOT	
	XOR	((S (B C (((B (C B NAND)) S) (((B (C B NAND)) S) (B C ((B (B NAND)) NAND)))))) I) $\rightarrow$ ((S (B C (E_2 (E_2 (B C ((B (B NAND)) NAND)))))) I)	

Figure 2-10: Cached expressions learned in the Boolean circuit experiment. Each expressions is shown with its name, if interpretable as a standard Boolean connective, and a schematic circuit showing its representation as a circuit. a) The three primitive logic gates used in the sampling procedure for the data set. b) Two higher-order learned expressions. These are used in c) to define TRUE, FALSE, and XOR.

This experiment is particularly suitable for inspecting the elements of the grammar that EC learns. Ideally, the algorithm recovers the constituent logic gates that were used to build up the tasks. Table 2-10 shows a few of the top ranked expressions in the library of the grammar referred to in Figure 2-9. These include the basic logic gates; we also show two higher-order expression E_1 and E_2, to stress that the representation the EC algorithm is using allows it to build concepts that are more expressive that just sub-circuits.

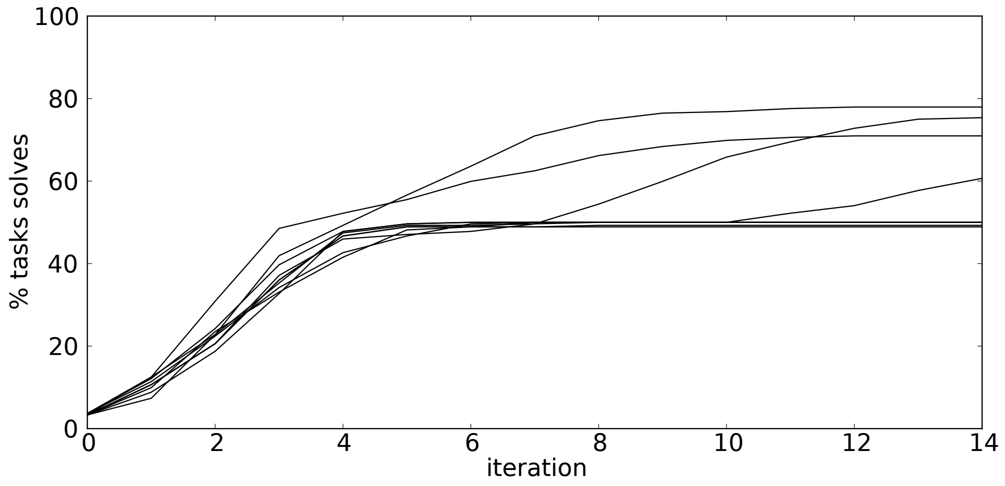


Figure 2-11: Learning curves from Boolean function learning Experiment 2. 10 learning curves for frontier size of 2000 on Z_{all}, a task set consisting of all Boolean functions of cardinality 3 or smaller. Note how most of the trajectories get stuck in a local minimum around 50%.

Experiment 2: An unstructured set of Boolean functions In this second experiment, we use task set Z_{all}, which corresponds to all 272 Boolean truth tables with three or fewer inputs as tasks. Since there is no implicit structure in this space of tasks, we expect this to be a more challenging task for EC. There are two kinds of learning that the EC algorithm might accomplish: first, many expressions in $\mathbb{L}$ map to a single Boolean function, so it needs to learn primitives that allow it to span many different functions instead of generating very simple functions redundantly. The second kind of learning involves the distribution of the functions themselves. In the circuit based Boolean experiment, that structure is apparent in the distribution in Figure 2-9. In this second experiment, we remove the second kind of structure. In Figure 2-11, we show 10 runs of the EC algorithm on the second experiment with a frontier size of 2000: note how there are several local minima that the majority of the runs get stuck in with performance around 50%, but several of the runs seem to take different trajectories to more successful representations.

Learning string transformations

String transformations are a class of problems encountered in the study of automation in spreadsheets (Gulwani 2011). In a string transformation problem, the learner is given a few examples of input-output string pairs and is asked to generalized to instances of input strings. Microsoft Excel's FlashFill implements program induction for solving such problems using a model that assumes a different generative model than EC: there is no use of a latent grammar, and individual string transformation problems are treated in isolation. However, many string transformation tasks have the potential for common underlying structure, making it a natural domain in which to apply the EC algorithm. Given a set of string transformation tasks that vary from simple foundational concepts to more complex concepts that compose on those simpler ones, EC is able to iteratively learn a sophisticated grammar that is expressive for those tasks.

We selected a set of relevant primitives for string transformation, listed in Table 2.1, and we created a small dataset of 20 string transformation problems of varying complexity, a sample of which is shown in Table 2.2a. Running EC on these tasks yields a learned grammar of program fragments that capture common building-blocks of such programs; these include a fragment which capitalizes each word in a string, a fragment which trims a string after a character, and fragments for the numbers -1 and 1. Table 2.2b shows some of the solutions that EC discovered; to be clear, here as elsewhere, we used the combinatory logic as our formalism, but the resulting programs are much more readable when translated to the lambda-calculus. Learning performance parameterized by number of iterations and frontier size is shown in Figure 2-12.

<ascii chars>...	char
zero	int
empty	str
string-of-char	char → str
string-of-int	int → str
upper	str → str
lower	str → str
capitalize	str → str
replace-substr-first	str → str → str
replace-substr-all	str → str → str
incr	int → int
decr	int → int
word-count	str → int
char-count	str → int
find-char	char → str → int
substr	int → int → str → str
replace	str → int → int → str → str
nth	int → str → str
fnth	(str → str) → int → str → str
feach	(str → str) → str → str
is	str → str → bool
filter-words	(str → bool) → str → str

Table 2.1: Primitives for string transformation given to the EC algorithm, and their associated types.

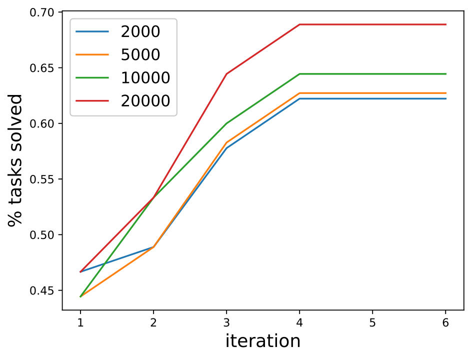


Figure 2-12: Learning curves as a function of frontier size. Larger frontier sizes help EC solve more tasks.

Conclusion

The case-studies above explore how vanilla EC behaves as the algorithm's frontier size is extended, as the number of iterations is increased, and as the amount of the modular structure is present in the problem domain. The results above suggest that if the frontier-size is sufficiently large, the grammars learned on each subsequent iteration of the algorithm is able to overcome the exponential scaling of the search

Task	*Input*	*Output*
1	Structure and Interpretation of Computer Programs	SICP
2	#include <os.h>	OS
3	ruby.clinton@mit.edu	Ruby Clinton

(a)

Task	*Solution*
1	λa.replace-substr-all empty (string-of-char ‘ ’) (feach (substr zero (incr zero)) (filter-words (λb.is b (cap b)) a))
2	λa.upper (substr (incr (findchar ‘<’ a)) (decr zero) (substr zero (findchar ‘.’ a) a))
3	λa.feach capitalize (replace-substr-all (string-of-char ‘ ’) (string-of-char ‘.’) (substr zero (findchar ‘@’ a) a))

(b)

Table 2.2: (a) String transformation tasks that EC learns to solve by composing solution fragments from simpler tasks. (b) The learned solutions to those tasks, translated from combinatory logic to lambda calculus for increased interpretability.

space.

2.6.2 Applications of Sequential EC

While the previous case studies examined the behavior of the EC algorithm in idealized learning settings, we present, in this section, two applications of Sequential EC in domains which intuitively fit the incremental and constructive nature of this variant of the EC algorithm.

Learning to Construct Graphical Models

This first case-study explores how EC could be used to aid human experts in the construction of statistical models. Graphical models – graphs that capture the dependency structure of a given statistical model – are central to probabilistic AI and Machine Learning, and learning the structure of graphical models directly from data is an active area of research (Adams, Wallach, and Ghahramani 2010; Tsamardinos, Brown, and Aliferis 2006; Friedman and Koller 2003; Heckerman, Geiger, and Chickering 2013). Graphical model structures enforce the conditional independencies a modeler believes exists among latent and observed random variables (Pearl 1989). In practice, the sorts of graphical models that humans design – Hidden Markov Models, phylogenetic trees, topic models, Ising models – exhibit certain symmetries and recursive structures that might be amenable to synthesis by programs.

A learner that is able to construct graphical models that fit a given dataset must be able to jointly generate hypothetical graphs and evaluate those graphs against the statistical relationships present in the data. Solving either of these problems on their own is quite difficult, and as we are primarily interested here in applying EC to domains with a rich compositional structure, we chose in this case-study to focus on the first part of this problem, i.e., learning the compositional structure underlying commonly used graphical models.

To do this, we constructed a set of tasks in which each task is a function from a list of nodes to a graph. That is, each task corresponds to a class of graphical models. See Figure 2-13 for the set of tasks and an example input-output pair for the function corresponding to each task.

In additional to the primitive combinators I, S, B, C, the base grammar for this case study also included the graph combinators described schematically in Figure 2-14. These graph combinators implement natural ways of composing two graphs together to create a third.

Our findings are that in initial iterations of the algorithm, Sequential EC is unable to find programs for the more complicated graph structures such as the Ising Model or the Hidden Markov Model; this is true even after enumerating ≈ 4 million programs in the frontier. However, it is able to find programs that generate simpler graphical models, such as Markov chains. By compressing the solutions to these simpler problems, it modifies its search procedure and, in subsequent iterations, learns more successful programs. An example of such a program shown in Figure 2-15, in which a

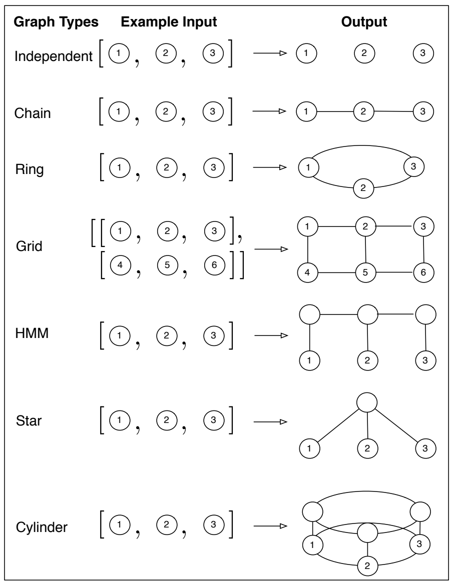


Figure 2-13: Example input/output pairs of learned programs. Each induced program takes as input an arbitrarily large list of observed nodes and produces an undirected graphical model. Labeled/unlabeled nodes correspond to observed/latent variables.

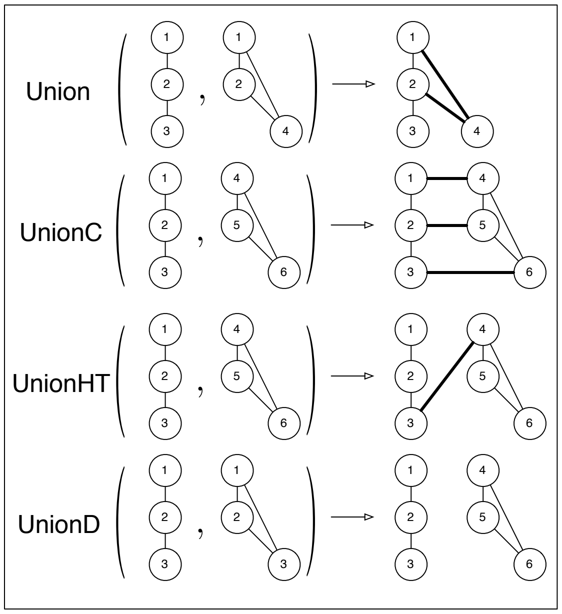


Figure 2-14: Graph combinators in initial library. Union: computes union of vertexes and edges. UnionC: adds edges between corresponding vertexes. UnionHT: adds edge between first (head) vertex and last (tail) vertex. UnionD: relabels input graph vertices so that they do not overlap and then computes union.

cylinder graph is generated. This example shows how relatively small compositional programs can be chained by Sequential EC to create programs of substantial size.

What grammars does Sequential EC learn, and how do they enable the bootstrap learning of more complicated programs? The simplest example is how the Markov chain bootstraps the learning of the Ising model. After one iteration, the program fragment (FOLDR UNIONC), used in the Markov chain program, is incorporated in to the grammar. On the second iteration, the Ising model is learned using this fragment, producing the program (((**S B**) MAP) ((FOLDR UNIONC) NULL)).

Building stable towers of blocks

Planning problems are a natural fit for Sequential EC, in that plans are sequences of steps towards a goal. Sequential EC allows for an approach to planning in which the building blocks of the plan are programs themselves and not just primitive actions. To investigate this perspective on Sequential EC, we applied EC to a classical AI planning problem; we show how the use of a functional programming language as the knowledge representation for the space of plans allows us to discover higher-order operations that mean, for example, "take a plan fragment and perform it twice in succession."

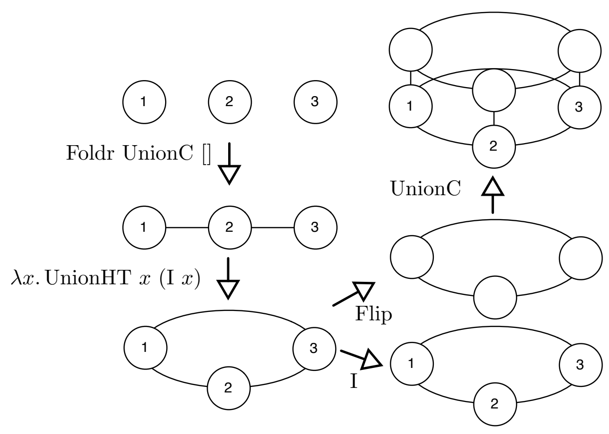


Figure 2-15: This program constructs a cylindrical structure from nodes by generating two rings – one latent, one visible – and connecting them. Sequential EC discovers this program by first discovering a program for generating rings.

The domain here is somewhat similar to the classic blocks world domain, in which agents must move stacked blocks on a tabletop from one configuration to another world (Winograd 1972). A key difference, however, is that in order domain the the goal is not to place blocks in any given configuration; rather, the goal is to construct structures that are both tall and stable. The learning agent begins with an empty tabletop, and has access to an unbounded supply of rectangular blocks that are either vertically or horizontally oriented. The learner's objective is to place the blocks in a configuration that maximizes the product of the structure height and its stability. The stability of a structure is operationalized as the fraction of "pushes" δ that the structure can withstand, where the set of such pushes ranges over a finite set of small pushes to the left and right (we represent the domain two-dimensionally). We compute the stability of a structure by simulating the pushes in the Bullet physics simulator[4]. Thus, the tasks in this domain are identical and are all of the form:

$$p(z|e) \propto \exp\left(\text{height}(T) \times \sum_{\delta} \text{stable}(T, \delta)\right) \tag{2.35}$$

In addition to the primitive routing combinators I, S, B, and C, the base grammar for this case study included primitives for placing horizontal and vertical blocks along the one-dimensional horizontal axis (given a block and its horizontal location, its final location is determined by lowering it – tetris like – until it comes to rest on either the ground or the block below it). Located blocks are represented as pairs of real numbers and Boolean values, where the Boolean value represents determines whether a block is horizontally or vertically oriented. Therefore, a program for constructed a tower of blocks is a list of pairs of real-values and Booleans. For example, in the first expression in Figure 2-16a, the subexpression `((pair -1) False)` denotes the pair $(-1, \texttt{False})$ and represents a horizontal block at location -1.

[4]Bullet: `www.bulletphysics.org`

```
Program :: [(Real,Boolean)]
```

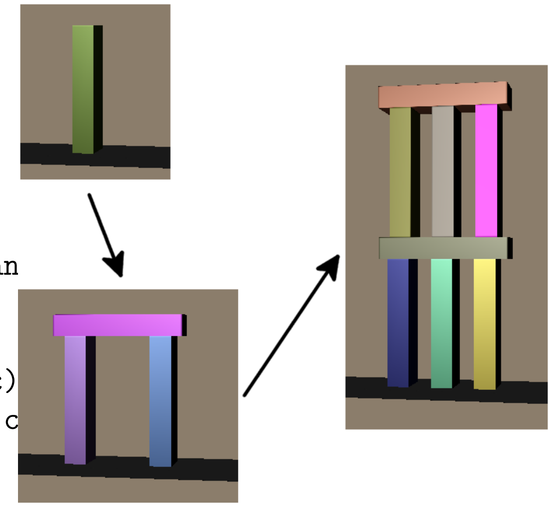
Program primitives:

```
cons :: a -> [a] -> [a]
append :: [a] -> [a] -> [a]
True,False :: Boolean
0,1,-1 :: Real
+,-,/,* :: Real -> Real -> Real
pair :: a -> b -> (a,b)
map :: (a -> b) -> [a] -> [b]
reverse :: [a] -> [a]
singleton :: a -> [a]
nand :: Boolean -> Boolean -> Boolean
first :: (a,b) -> a
second :: (a,b) -> b
toFirst :: (a -> b) -> (a,c) -> (b,c)
toSecond :: (a -> b) -> (b,a) -> (b,c
```

Subset of learned grammar:

```
/* place a horizontal block */
(cons ((pair -1) False))
/* place a vertical block */
(cons ((pair 1) True))
/* duplicate a tower-building plan */
((S append) I)
```

(a)



(b)

Figure 2-16: Using Sequential EC to learn to build tall and stable towers.

We used the Sequential EC algorithm to incrementally build up towers while focusing our search effort on the most promising tower building plans. A subset of the useful expressions that Sequential EC learns are shown in Figure 2-16a.

We find that Sequential EC learns to build tall yet stable towers by first learning program fragments that put down individual blocks, then, after rounds of compression, composing those fragments to build multi-block towers and learning a program that duplicates tower-building plans, allowing it to make multilevel towers.

Conclusion

Sequential EC is a way to combine the goal-directed nature of planning with the rich compositional structures afforded by EC. In this section, we showed how we applied Sequential EC to two domains that seem to be a natural fit for this combination.

2.6.3 Applications of Symbolic Dimensionality Reduction

In this section, we show how the symbolic dimensionality reduction version of EC can move beyond mere *learning* and towards *understanding.* In the first of the case studies in this section, we apply EC to multitask symbolic regression, where the learner's goal is to uncover the algebraic form that represents a domains "natural law." In the second case study, we apply the same algorithm to the problem of learning linguistic morphology, a domain in which people intuitively understand morphological transformations as the applications of "rules."

Multitask regression

In this case study we explore how the symbolic dimensionality reduction version of EC (see Section 2.5.3) performs on a version of symbolic regression task in which the underlying domain is a mixture of function classes.

Scientists classify natural laws according to the structure of the equations that describe them. For example, *inverse square laws* describe both gravitation (Newtonian gravity) and electrostatic forces (Coulomb's law); *power laws* describe both the motion of planets (Kepler's laws) and the distribution of words (Zipf's law); and *polynomials* arise in both classical mechanics (e.g., the trajectory of a object when thrown) and in geometry (e.g., the equation of a line). In all these examples, scientists induced a new abstraction from particular examples of functions obeying that abstraction.

Here we model the learning of high-level abstractions (*power law*, *linear relationship*, etc.) as multitask regression. We present EC with a number of curve-fitting tasks, $\{z_i\}_{i=1}^N$. For each curve, written $\{f_i(\cdot)\}_{i=1}^N$, the model observes noisy samples of the function's value on a fixed set of points, written X. This framing gives our definition of the tasks:

$$z_i(e_i) = \prod_{x \in X} \text{NormalDensity}(e_i(x) | \mu = f_i(x), \sigma = 1) \tag{2.36}$$

Here, the tasks have a natural interpretation as a likelihood model. If one assumes that Gaussian noise has been added to observed values of $f_i(x)$ for $x \in X$, then Equation 2.36 is the likelihood of e_i's predictions on X.

Running EC on a set of related regression tasks yields a mixture of program pieces that explain the curves and abstract out commonalities. However, this mixture lacks the crisp interpretability of a concept like *second order polynomial* or *Fourier basis.* To recover concepts like these we induce a single program using symbolic dimensionality reduction (Section 2.5.3).

The domain of problems and the learned results for this case study is shown in Table 2-17. Each row represents a different class of functions, described in the first column. For example, the first row represents the class of lines, while the third is the class of functions that is the composition of either a *sin*, *cos* or *squaring* function with a line that goes through the origin. The second column shows examples of data provided to EC: for each function in the corresponding class, input values where

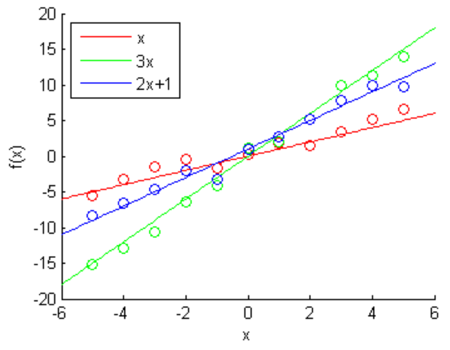
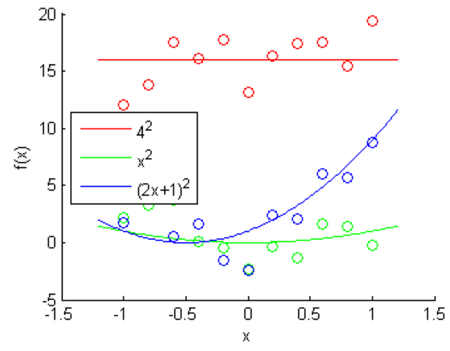
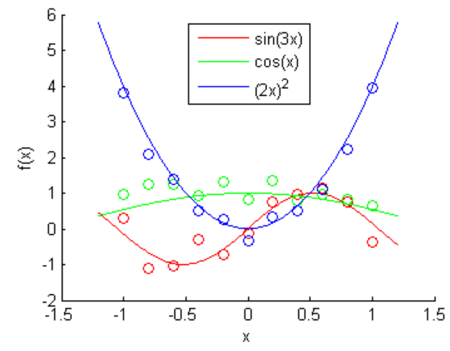
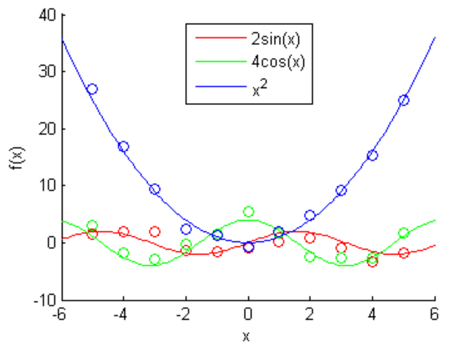
$f(x) = \ldots$	Example input data	Learned Expression
$ax+b$ where: $a \in \{0,1,...,9\}$, $b \in \{0,1,...,9\}$		$\lambda abx.ax+b$
$(ax+b)^2$ where: $a \in \{0,1,...,9\}$, $b \in \{0,1,...,9\}$		$\lambda abx.(ax+b)^2$
$g(ax)$ where: $a \in \{0,1,...,9\}$, $g \in \{\sin, \cos, \text{square}\}$		$\lambda gax.g(a \times x)$
$a \times g(x)$ where: $a \in \{0,1,...,9\}$, $g \in \{\sin, \cos, \text{square}\}$		$\lambda gax.a \times g(x)$

Figure 2-17

/kæts/ (“cats”)	/dɔrz/ (“doors”)
/ʃuz/ (“shoes”)	/bʊks/ (“books”)
/hɔrsəz/ (“horses”)	/ajz/ (“eyes”)

Figure 2-18: Plural forms of nouns commonly spoken by children at thirty months of age (Dale and Fenson 1996), written in phonetic (IPA) form.

```
(lambda (stem)
 (append stem (if (voiced? (last-one stem))
                  ((cons /z/) null)
                  ((cons /s/) null))))
```

Figure 2-19: Learned representation of the English plural. When given the stem of a word (like “dog:” /dag/) this expression produces a plural form (like “dogs”: /dagz/).

past tense:

```
(lambda (stem)
 (append stem
  (if (voiced?
   (last-one stem))
   ((cons /d/) null)
  ((cons /t/) null))))
```

superlative:

```
(lambda (stem)
 (append stem
  (cons /@/
  (cons /s/
  (cons /t/ null)))))
```

comparative:

```
(lambda (stem)
 (append stem
  (cons /@/
  (cons /r/ null))))
```

Figure 2-20: Other learned representations of English inflections.

chosen at regular intervals; output values were perturbed to contain a small amount of noise.

As shown in the first two rows of Table 2-17, EC can recover abstractions like *linear relationship* or *squared polynomial*. However, these abstractions do not employ the sophisticated machinery afforded by an expressive LOT — a standard symbolic regression toolkit could also solve these problems. The next two rows of Table 2-17 show the system learning abstractions that take other functions as parameters, such as scaling an arbitrary function by some real number. We see this as an advantage of using rich representations like combinatory logic.

English morphophonology

Consider a problem that nearly every child faces: that of acquiring the linguistic rules of his or her native language. During the first few years of life, children learn their languageâĂŹs rules for forming plurals, past tense, superlatives, past participles, and other forms of inflectional morphology (O’Donnell 2015). Although forming the English plural may seem simple to a native speaker, the regular rule actually consists of three different cases that the child must learn; depending upon phonetic properties of the end of the noun, a different sound is appended to it. Additionally, the child must identify a large set of irregular patterns, such as *ox→oxen*. When confronted with linguistic data such as in Figure 2-18, children can, and machines ought to, explain this data by inducing the general morphophonological rule of the English plural.

We presented EC with sets of words all sharing the same inflection, for example all being plural words or all being superlatives. We then asked the system to discover programs that evaluate to the pronunciations of these words. So in this setting each task is a word, and our likelihood model for task z_w for word w is:

$$p(z_w|e) = \begin{cases} 1, \text{ if } e \text{ evaluates to } t \\ 0, \text{ otherwise.} \end{cases} \tag{2.37}$$

English possesses eight classes of inflectional affixes: the plural, past tense, past participle, genitive, 3rd person singular, progressive, comparative, and superlative. For each inflectional affix, we compiled a word list, drawn from (Dale and Fenson 1996), of words modified to use that affix. Each word was represented as a sequence of phonemes. For example, when learning the plural, the system is given a set of words like in Figure 2-18, or when learning the superlative, the system is given the pronunciations of words like *biggest*, *tallest*, or *smallest*.

Like in multitask regression (Section 2.6.3), our goal is not to estimate a distribution over programs, but to induce a single program that compresses the observations. So, as in multitask regression, we use the symbolic dimensionality reduction variant of EC. But not all words conform to regular grammatical rules: for example the plural of "tooth" is "teeth" and not "tooths". So we introduce an extra parameter, ϵ, which corresponds to the probability of a word not following the regular grammatical rule. We seek the single program d^* maximizing:

$$d^* = \arg\min_d \min_M \left(-\log P_{\cdot|G}(d|G_0) - \sum_i \log \left((1-M) \sum_{\substack{e_i: \\ d(e_i)=x_i}} P_{\cdot|G}(e_i|G_0) + M \sum_{\substack{e_i: \\ e_i=x_i}} P_{\cdot|G}(e_i|G_0) \right) \right) \tag{2.38}$$

Some inflectional rules, such as that of the superlative, modify all (regular) stems in the same manner. In these cases, our system recovered the correct regular inflection. For the superlative, this inflection consists of appending /əst/ to the stem.

2.7 Conclusion & Future Directions

In this paper, we presented an algorithm for multi-task functional program synthesis that automatically learns the structure of the search space by discovering useful reusable program components. We motivated the approach and algorithm by framing the problem as approximate posterior inference in a hierarchical probabilistic model, studied the algorithm's behavior on several domains, and showed how it can be used in various applications. Several issues remain to be tackled before the synthesis of functional programs will be a practical approach to solving difficult AI search problems. As we discuss below, some of these issues are representational. But even when issues of representation are solved, issues of scaling will remain.

Knowledge representation Our use of the combinatory logic as our computational formalism was motivated by its simplicity and lack of explicit variable binding.

Unfortunately, the lack of variable binding quickly leads to large expressions that do nothing but route input arguments to their positions in the combinatory expression. And these large expressions can quickly overwhelm any reasonably sized search frontier. It is unclear to what extent learning of reusable combinators can ameliorate this problem.

This problem is related to the more general problem of how to deal with state in functional programs. The practical use of real-world functional languages usually requires special syntactic mechanisms to thread state through complex programs. We would expect that if these mechanisms are necessary to manage the complexity of the code that humans write, then they will also be very useful for program synthesis.

Continuous domains Many domains involve learning domains are most parsimoniously represented as containing continuous dimensions. Learning these parameters as part of a search over program structure is bound to be very inefficient, because such a search does not take advantage of gradient information. Future work must be done to understand the interaction between learning continuous parameters, discrete parameters, and program structure.

Non-deterministic or probabilistic programs Although EC frames program synthesis as inference in a probabilistic model, the programs that it synthesizes are themselves deterministic. In many contexts, however, the knowledge we want programs to represent is itself non-deterministic or probabilistic in nature. We have presented one approach to tackling this problem in the section on symbolic dimensionality with EC, but it is not clear whether all such knowledge should be handled in this way. That is, it is not clear whether it is best to handle all non-determinism as non-determinism over the programs themselves, or whether non-deterministic constructs shouldn't be included in the program formalism itself.

Chapter 3

The Infinite Knowledge Base Model

3.1 Introduction

People routinely learn and reason about complex inter-related systems of concepts. Young children learn about kinship relations (MOTHER, DAUGHTER, COUSIN, ...), ontologies of living things (ANIMAL, PLANT, PET, PREDATOR, ...), and social rules (MINE, YOURS, SHARED, BORROWED, ...). The basic conceptual ontology of space and time as well as highly-specialized systems of technical concepts (such as those of Newtonian mechanics, FORCE, MASS, ACCELERATION, VELOCITY, ...) also fall into this category. Arguments and evidence from philosophy, psychology, linguistics and AI support the view that our conceptual knowledge is best represented in a symbolic and compositional formalism – most often based on the first-order predicate calculus – which is often termed the Language of Thought (LOT) (Fodor 1975; Fodor and Pylyshyn 1988). If our goal is human-like AI, then we need learning algorithms that can acquire these language-like representations from data.

As a motivating example, consider a small child who at home and at school witnesses various relationships between individuals. Among these social relations are "mommies," "daddies," "brothers, " "sisters," "daughters" and "sons." But she may never or rarely hear about the concepts that underlie these relations: "male," "female," "parent," "child," "sibling," "spouse," "relative," "family," to name a few. Yet we would hardly say a child understands what a mother is if not as a female parent or what a brother is if not as the different son of the same parent.

Our intuition is that in order to obtain an understanding of these relations, the child must be continually searching for rules that *explain* why the observed relations, and not others, are the ones she sees. In doing so, she attempts to strike a balance between the complexity of the rules and the precision with which they predict the data. Although we will not attempt here to address this learning task in its full complexity (after all, it takes children many years to master the kinship relations), this is the kind of knowledge acquisition problem our approach here is aimed at tackling. In the case studies below, we will focus on a variant of the kinship domain and various relations defined on small graphs.

We approach this learning challenge with two main ingredients. First, we use the

Meta-Interpretative Learning (MIL) framework, introduced by Muggleton and Lin 2013. MIL is a top-down approach to *Inductive Logic Programming* (ILP) (Muggleton and Raedt 1994; Muggleton et al. 2012) which flexibily allows for predicate invention, the ability to use new symbols denote intermediate or abstract concepts. The second ingredient is the idea of concept learning as Bayesian inference, which enables us to characterize concept acquisition as probabilistic inference in a generative model over expressions in a LOT (Piantadosi, Tenenbaum, and Goodman 2012; Piantadosi, Tenenbaum, and Goodman 2010; Dechter et al. 2013; Kemp, Goodman, and Tenenbaum 2008; Kemp, Goodman, and Tenenbaum 2007).

As with most work in ILP, we use datalog (a fragment of definite clause logic) as our LOT, and show how to specify a probabilistic generative model over datalog knowledge bases. Finally, we present a principled MAP inference algorithm for this generative model that uses ideas from state-space heuristic search and classical planning to guide learning in the resulting hypothesis space.

Specifically, the main contributions of this paper are:

- we introduce the Infinite Knowledge Base Model (IKM), a probabilistic generative model over first-order relational knowledge bases;
- we a describe the HS-MAP(IKM) algorithm, a MAP inference algorithm for the IKM that guides search using an automatically constructed domain-agnostic heuristic;
- and we empirically explore the behavior of this model and algorithm on a number of domains.

The remainder of this paper is structured as follows. After briefly discussing related work and presenting some preliminary concepts and notation, we describe the IKM in Section 3.4. Subsequently, in Section 3.5, we describe the HS-MAP-IKM algorithm. In Section 3.6, we study the HS-MAP-IKM algorithm empirically. We finish with a discussion of the work presented here, its limitations, and future work to handle these limitations and expand its applicability and scope.

3.2 Related work

The work presented here is partially inspired by the *Infinite Relational Model* (IKM) (see Kemp et al. 2006). The IKM is a hierarchical clustering model of relational concept learning. In this model the propensity of a tuple $(x_1, \ldots, x_n)$ to be a member of an observed relation R is determined by the latent clusters to which $x_1, \ldots, x_n$ belong. The cluster assignments for objects are distributed according to a CRP, promoting concept definitions that use a smaller number of latent clusters. In the case of learning multiple concepts at once, the latent clusters are shared across concept definitions. Thus, like the IKM, the IKM uses a non-parametric probabilistic generative model to guide concept learning with an unbounded number of latent concepts. However, the hypothesis space of concept definitions in the IKM, being defined as a clustering model over individual objects, is quite restrictive. The IKM attempts to extent this kind of probabilistic generative model to a more expressive relational logic.

In the ILP literature, Muggleton 1997 introduced the idea of learning a logical theory from positive data with the aid of a probability distribution over theories and the data they generate. In that work, the probability of a theory was used to guide the Progol algorithm, which uses a bottom up *inverse entailment* approach. As discussed in Muggleton and Lin 2013, such bottom-up approaches appear not to lend themselves to predicate invention. Muggleton 1997 did not propose a specific generative model over logical theories, instead using the number of atoms in the theory as a proxy for negative log probability of the logical theory.

3.3 Preliminaries

As mentioned above, this work is based on the MIL approach to ILP. As background, we introduce MIL and the associated concepts and notation we will adopt here.

ILP is a symbolic machine learning approach whose goal is to learn logic programs that parsimoniously explain a data set. Logic programs are collections of clauses that together define one or more predicate, and an intepreter for a logic program is a program that answers queries about the predicates the logic program defines. A *meta-interpreter* is a predicate defined in a logic program that simulates the behavior of an interpreter, but usually in such a way that it modifies the default behavior of the underlying interpreter.

MIL uses a meta-interpreter to facilitate efficient search and predicate invention when inducing logic programs from data. The basic idea is that the meta-interpreter has access to a set of *metarules*, higher-order definite clauses that function as templates for first-order definite clauses. A metarule has one or more *metavariables* in it, and the metarule is instantiated by binding these metavariables to symbols. While searching for a proof of a query, the MIL meta-interpreter instantiates these metavariables. In doing so the meta-intepretator implicitly generates a first-order logic program that entails the data. By allowing an unbounded supply of symbols for instantiating metavariables, this scheme allows the learner to invent predicates that have occurred neither in the background knowledge nor the data.

Muggleton and colleagues introduced METAGOL$_D$, an algorithm and implementation of the MIL framework, along with several variants based on it. To induce a logic program, METAGOL$_D$ performs an *iterative deepening* search (Korf 1985a) on the size of the induced logic program; this approach guarantees that the learner discover the shortest logic program that entails the positively labeled data, without entailing any negatively labeled data.

While METAGOL$_D$'s objective is to find shortest logic program that entails positively labeled data and does not entail negatively labeled data, a slightly different algorithm METAGOL$_O$ minimizes the *resource complexity* of the hypothesized logic program instead. METAGOL$_O$ modifies METAGOL$_D$ in two ways: first, the background knowledge is annotated with the the cost of using each rule. Second, the iterative deepening meta-intepretator accumulates this cost for each proof, maintaining an upperbound on the lowest cost proof. Thus, once METAGOL$_O$ finds one solution (necessarily the same one found by METAGOL$_D$), it continues the search, pruning those partial proofs that exceed the current upper bound on the cost.

Although METAGOL$_O$ allows for more flexibility in the learner's objective function, the cost of a partial solution is used only to prune hypotheses, not to guide the search. The approach we present in this paper defines the cost of a partial solution to be the negative logarithm of a probability mass function defined by the probabilistic generative model introduced in the next section. Our search algorithm uses this objective function in a more explicit way to direct the search over abductive proofs. Later in Section 3.6.4, we compare our approach to METAGOL$_O$ empirically.

Notation and definitions The following are some of the logic programming concepts and notational conventions we will be using.

A variable is represented by a string of letters and numbers whose first character is an upper case letter (e.g. $X, Y, \ldots$). A predicate or function symbol is string of letters of numbers whose first character is a lower case letter (e.g. $p, q1, \ldots$). The arity $\mathrm{ar}(p)$ of a predicate or function symbol p is the number of arguments that p takes. A function symbol a is a constant if $\mathrm{ar}(p) = 0$. A *term* is a constant, a variable, or a function symbol applied to a tuple of terms, the function's *arguments* (e.g. X, a, and $f(X, a)$ are terms). A variable is a *first-order* variable if it can be substituted for by a term; it is a *higher-order* variable if it can be subsituted for by a predicate symbol. An *atom* is a predicate or higher-order variable followed by a tuple of terms, where the length of the tuple is equal to the arity of the predicate or higher-order variable; it is a *datalog atom* if all the its terms are constants or variables (e.g. $f(X, 1)$ is a datalog atom but $f(f(X), 1)$ is not). A formula is *ground* if it contains no variables. A *datalog clause* C has the form $H \leftarrow B$ where H, a datalog atom, is the *head* of the C and B, a finite conjunction of datalog atoms, is its *body*. A *datalog clause* is *(first-order) range-restricted* if all (first-order) variables that appear in the head also appear in the body. In this paper, all clauses will be required to be range-restricted. The body of a clause may be empty, in which case the clause is called a *fact*. A *first-order atom* is an atom that contains only first-order variables. A *higher-order atom* is an atom that contains higher-order variables. Likewise, a *first-order clause (higher-order clause)* is a clause that contains first-order (higher-order) atoms.

A *knowledge base* K is a set of first-order clauses. For the standard semantics of first-order datalog knowledge bases, see Abiteboul, Hull, and Vianu 1995. The *extension* $E(K)$ of a first-order knowledge K is the set of ground atoms entailed by the K. A *substitution* $\Phi = \{V_1 \mapsto c_1, V_2 \mapsto c_2, \ldots\}$ is a partial function from variables V to terms c. A substitution Φ applied to a term or formula t replaces all occurences of V with c with $\Phi(V) = v$. We call a substitution *ground* if its range consists of ground terms.

The MIL framework introduces a few additional concepts: a *metarule* r is a formula of the form $\exists \mathbf{V}.C$ where C is a higher-order datalog clause, and $\mathbf{V}$ is some subset of the variables occuring in C. We call $\mathbf{V}$ the *metavariables* of r. We require that for metarule r to be well-formed, all of the higher-order variables in C must be contained in $\mathbf{V}$; note, however, that we do not require that every variable in $\mathbf{V}$ be higher-order.

3.4 The Infinite Knowledge Base Model

The IKM is a generative model that, given a *knowledge base schema* Σ, specifies a joint probability distribution over knowledge bases K and the observed data sets $\mathcal{D}$ that are generated from them. In broad terms, the generative model over a knowledge base and a dataset is described by the following sampling procedure:

- sample the number of clauses N that will be in the knowledge base;
- for each of the N clauses in knowledge base, sample an associated metarule from the knowledge base schema;
- for each metavariable occuring in the knowledge base, bind it to a symbol drawn from a distribution over symbols;
- the N metarules and the associated metavariable bindings defines a first-order datalog knowledge base; compute the set of observable facts entailed by this knowledge base;
- from this set of observable facts, sample a subset to be the data set.

The form of the distributions mentioned this description along, with their parameterizations, will be described in the following sections.

3.4.1 Knowledge base schema

The IKM defines a hypothesis space $\mathbf{H} = \mathbf{H}(\Sigma)$ over first-order knowledge bases that is parameterized by a *knowledge base schema* Σ. A knowledge base schema $\Sigma = (R, \mathbf{S})$ is a pair in which R is a set of metarules, and where each $\mathbf{S}^{(i)}$ in $\mathbf{S} = \{\mathbf{S}^{(1)}, \mathbf{S}^{(2)}, \ldots,\}$ is a potentially infinite ordered set of symbols of arity i.

A symbol s in $\mathbf{S}^{(i)}$ is either a *named* or an *unnamed* symbol. A symbol is named when it occurs "in the world" (i.e. in the data set or in some background knowledge the learner may have). All other symbols are unnamed. Unnamed symbols are arbitrary in the sense that if P is a permutation of the symbols in $\mathbf{S}^{(i)}$ that leaves the named symbols in the same place, then the meaning of a knowledge base K does not change if we simultaneously replace every occurence of every symbol $s \in \mathbf{S}^{(i)}$ with $P(s)$. In this paper, we will use the symbols $\ell_j^{(i)}$ as teh arbitrary name of the unnamed symbol whose arity is i and whose index in $\mathbf{S}^{(i)}$ is j. When there is no ambiguity, we will drop the arity and just write ℓ_j. In the context of learning, we will refer to unnamed symbols as *latent symbols* — and the associated predicates as *latent predicates* — to highlight that they represent predicates or objects that the learner does not directly observe.

Example 1. $\Sigma = (R, \mathbf{S})$ *is a knowledge base schema where*

$$R = \left\{ \begin{array}{l} r_1 = \exists ABC.A(X,Y) \leftarrow B(X), C(Y) \\ r_2 = \exists AB.A(B) \end{array} \right\},$$

and

$$\begin{aligned} \mathbf{S}^{(0)} &= \{1, 2, 3, 4, \ell_1^{(0)}, \ell_2^{(0)}, \dots\} \\ \mathbf{S}^{(1)} &= \{a, \ell_1^{(1)}, \ell_2^{(1)}, \dots\} \\ \mathbf{S}^{(2)} &= \{q, \ell_1^{(2)}, \ell_2^{(2)}, \dots\}. \end{aligned}$$

This knowledge base schema consists of two metarules r_1, r_2 and six named symbols $1, 2, 3, 4, a, q$. The digits are constants, a is a unary predicate symbol, and q is a binary predicate symbol. In addition, S contains an infinite number of unnamed constants, unary predicate symbols and binary predicate symbols.

A *metarule instantiation* $\delta = (r, \Phi)$ is a pair of a metarule r and a ground substitution Φ that binds the metavariables of r. We say that metarule instantiation $\delta = (r, \Phi)$ is *complete* if every metavariable of r is bound by Φ; otherwise, it is *incomplete*. A complete metarule instantiation corresponds to a first-order clause (although many metarule instantiations can result in the same first-order clause). We write $C = C(\delta) = C((\exists \mathbf{V}.C_0, \Phi))$ to mean that $C = \Phi(C_0)$.

Similarly, a *(knowledge base) schema instantiation* Δ is a list of metarule instantiations. If $\Delta = (\delta_1, \dots, \delta_N)$, then the knowledge base it corresponds to is $K = (C(\delta_1), \dots, C(\delta_N))$, and we write $K = K(\Delta)$. A schema instantiation is complete if all of its metarule instantiations are complete; otherwise, it is incomplete.

Example 2 (schema instantiation)**.** *Let Δ be an complete instantiation of the schema in Example 1 where:*

$$\Delta = \left\{ \begin{array}{l} \delta_1 = (r_1, [A \to a, B \to \ell_1, C \to \ell_2]) \\ \delta_2 = (r_2, [A \to \ell_1, B \to 1]) \\ \delta_3 = (r_2, [A \to \ell_1, B \to 2]) \\ \delta_4 = (r_2, [A \to \ell_2, B \to 3]) \\ \delta_5 = (r_2, [A \to \ell_2, B \to 4]) \end{array} \right\} \tag{3.1}$$

The knowledge base corresponding to this instantiation is

$$K(\Delta) = \left\{ \begin{array}{l} a(X,Y) \leftarrow \ell_1(X), \ell_2(Y) \\ \ell_1(1) \\ \ell_1(2) \\ \ell_2(3) \\ \ell_2(4). \end{array} \right\} \tag{3.2}$$

$K(\Delta)$ defines the extension of predicate $a/2$ to be $\{a(1,3),\ a(2,3),\ a(1,4),\ a(2,4)\}$.

3.4.2 The probability distribution

We can break up the probability distribution over data sets into the two standard components of the Bayesian inference framework, the prior over knowledge bases and the likelihood the knowledge base given the data (i.e., the probability of observing the data set given a specific knowledge base).

Prior distribution

The prior distribution over knowledge bases $p_{\text{prior}}(K \mid \Sigma, \Theta, \Delta_0)$ is parameterize by a knowledge base schema Σ, schema parameters Θ, and an initial schema instantiation Δ_0. The schema parameters are $\Theta = (\gamma, \alpha_R, \alpha_{\mathbf{S}})$, where $0 < \gamma < 1.0$, $\alpha_R > 0$ and for $a \in 0, 1, \ldots, \alpha_{\mathbf{S}^{(a)}} > 0$. We will refer to the pair (Σ, Θ) as a *parameterized schema*.

The stochastic process that defines $p_{\text{prior}}(K \mid \Sigma, \Theta, \Delta_0)$ is shown in Algorithm 4. To sample a knowledge base K given a parameterized schema $\Sigma(\Theta)$, we sample a schema instantiation Δ. First, we sample the number of metarules N in the instantiation from a geometric distribution with success probability γ. Next, we sample N metarules $\bar{r}$ from a Dirichlet-Multinomial distribution with parameter α_R. Finally, we sample a substitution that instantiates the metavariables in Δ. For each arity a present in the Δ, the metavariables V_a are jointly sampled from a Chinese Restaurant Process (Pitman 2006), the non-parameteric generalization of the Dirichlet-Multinomial; using a non-parametric distribution allows for the unbounded number of symbols in $\mathbf{S}$.

If there is background knowledge we want to include, as there usually is, it can be included in the intial schema instantiations. If the background knowledge contains no meta-variables, then we can ignore it and stipulate that it is contained in all knowledge base instantiations. On the other hand, if we want to include instantiated metarules in Δ_0, we can condition the Dirichlet-Multinomial and CRP on Δ_0, taking the corresponding rule and symbol counts into account. Since the Dirichlet-Multinomial and CRP are both conjugate to the schema instantiation, conditioning on the schema instantiation leaves the form of the distribution unchanged.

Algorithm 4: IKM prior distribution over a knowledge base instantiations

```
Data: $\Sigma, \Theta = (0 \geq \gamma \leq 1.0, \alpha_R, \{\alpha^{(a)} \geq 0\}_{a \in 1,2,\ldots})$: parameterized knowledge
      base schema
$\Delta_0$: initial schema instantiation
Result: $\Delta$: a complete schema instantiation
1 $N \sim \text{Geom}(\cdot|\gamma)$
2 $\bar{r} = (r_1, \ldots, r_N) \sim \text{Dirichlet-Multinoulli}(\cdot|\Sigma, \Delta_0, \alpha_R)$
  for $a \in 1, 2, \ldots$ do
3     $\Phi(V_a(\bar{r})) \sim \text{CRP}(\cdot|\Sigma, \Delta_0, \alpha^{(a)})$
  end
  $\Delta \leftarrow \Delta_0 \cup (\bar{r}, \Phi)$
  return $\Delta$
```

In Section 3.5, we will describe an algorithm for *maximum a posteriori* (MAP) inference over this distribution, i.e, the problem of searching for the highest posterior probability schema instantiation given some dataset. For the MAP solution to be a useful representative of the posterior distribution, it is important to account for symmetries within the space of samples generated by the generative process we have specified. We will say that two schema instantiations are *variants* if the corresponding knowledge bases are syntactically identical up to renaming of variables; we say that they are *equivalent* if the knowledge bases they correspond to have the same extension. The first source of symmetry is that if we take a schema instantiation, permute the order of its metarules, and then reindex the unnamed symbols so that their first appearance is in increasing order, the corresponding knowledge base is equivalent to the original one. This is because the extension of a datalog knowledge base is invariant to the order of its clauses. We will call this symmetry the *rule permutation symmetry*.

The rule permutation symmetry arises from the fact that applying a permutation to the rules in a schema instantiation, and then reindexing the unnamed symbols so that the schema instantiation is well-formed, results in a new schema instantiation with the same probability under our generative model; this is because a) the prior is exchangeable (as noted above), and because b) the extension of named symbols does not depend on the rule order or the indexing of the unnamed symbols.

Let Δ a schema instantion with $|\Delta|$ rules. We can account for the rule permutation symmetry, by multiplying the probability of sampling Δ under our generative model by $|\Delta|!$. One might be concerned, however, that in doing so we overcount. Define the $\mathcal{T}(\sigma, \Delta)$ to be the schema instantiation that results from permuting the rules of Δ by permutation σ and then reindexing the unnamed symbols in increasing order. There may be two schema instantations Δ_1 and σ such that $\Delta_2 = \mathcal{T}(\sigma, \Delta_1)$ is a variant of Δ_1. In this case, we do not want to count both instantiations towards the probability of the associated equivalence class. We will argue that we can avoid checking for this overcounting because in such cases Δ_1 always contains at least one redundant in clause (in the sense that if you removed it you the extension of the knowledge base remains unchanged). Since Δ_1 has a redundant clause, it cannot be an optimal solution, and the optimal solution remains unchanged if we remove all such knowledge bases with redunant clauses from the hypothesis space.

Proposition Let Δ_1 be a schema instantiation $\Delta_1 = (r_1, \Phi_1), \ldots, (r_N, \Phi_N)$. Let $\Delta_2 = \mathcal{T}(\sigma, \Delta_1)$ where σ is a permutation such that $K(\Delta_2)$ is a variant of $K(\Delta_1)$. Then there exist two clauses in c_1 and c_2 in $K = K(\Delta_1)$ such that $E(K) = E(K - \{c_1\}) = E(K - \{c_2\})$. That is, c_1 is redudant given c_2 and vice versa.

Proof Suppose permutation σ maps the ith metarule instantiation δ_1 in Δ_1 to the jth metarule instantiation δ_2 in Δ_2, where $i \neq j$. And let δ' be the ith metarule instantiation in Δ_2. Then by assumption, clause $c' = c(\delta')$ is a variant of clause $c_1 = c(\delta_1)$. Now, since $K(\Delta_1)$ and $K(\Delta_2)$ are variants of one another, the transformation $\Delta_2 = \mathcal{T}(\sigma, \Delta_1)$ induced some reindexing of unnamed symbols. Let $\mathcal{I}(\cdot)$ be the function that rewrites the unnamed symbols under this reindexing. Let s_1 be the set of proofs in $K(\Delta_1)$ that use clause c_1 and that prove facts of named predicate symbols. Then $s_2 = \mathcal{I}(s_1)$ is a set of proofs of the same facts in $K(\Delta_2)$, and all of the proofs in s_2 use clause $c_2 = \mathcal{I}(s_2)$. But since $K(\Delta_1)$ and $K(\Delta_2)$ are variants, s_1 is also a set of proofs for those same facts in $K(\Delta_2)$. Therefore, we can remove either of s_1 or s_2 from the set of possible proofs in $K(\Delta_2)$ without changing the extention of $K(\Delta_2)$. By construction, s_1 contains all the proofs that use clause $c_1 = c'$ and s_2 contains all the proofs that use c_2. Therefore, we can remove either of these clauses from $K(\Delta_2)$ without changing its extension. $\square$

Since a schema instantiation whose knowledge base has the same extension but fewer rules is always preferred by our generative model (because we of the geometric distribution over number of clause in the KB), any such schema instantiation is suboptimal.

Taking the above counting argument into account, we have the following prior probability distribution over schema instantiations:

$$\begin{aligned}
p_{\text{prior}}(K = K(\Delta) \mid \Sigma, \Theta) =& |K|!\, p_{\text{prior}}(\Delta \mid \Sigma, \Theta) && (3.3)\\
p_{\text{prior}}(\Delta \mid \Sigma, \Theta) =& \text{Geom}(|\Delta|; \gamma) && (3.4)\\
& \text{Dirichlet-Multinoulli}(\bar{r}|\alpha_r) && (3.5)\\
& \prod_a \text{CRP}(\{x | V \mapsto x \in \Phi(\Delta) \text{ s.t. } \text{ar}((V)) = a\} | \alpha^{(a)}) && (3.6)
\end{aligned}$$

The probability density for the CRP is

$$\text{CRP}(B; \alpha) = \frac{\Gamma(\alpha)}{\Gamma(\alpha + N)} \alpha^K \prod_k \Gamma(N_k), \tag{3.7}$$

where $B = B_1, \ldots, B_k$ is partition of the integers $1, \ldots, N$ and the number of elements in partition B_k is N_k.

The probability mass function for the symmetric Dirichlet-Multinoulli is:

$$p(\{N_k\}|\alpha) = \frac{\Gamma(K\alpha)}{\Gamma(N + K\alpha)} \frac{\prod_k \Gamma(N_k + \alpha)}{\Gamma(\alpha)^K} \tag{3.8}$$

where α is the concentration parameter, K is the number of categories, N_k is the number of items in category k, and $N = \sum_k N_k$ is the total number of items assigned. This gives a log probability of

$$\begin{aligned}
\log p(\{N_k\}|\alpha) =& \log \Gamma(K\alpha) - \log \Gamma(N + K\alpha) && (3.9)\\
& - K \log \Gamma(\alpha) + \sum_k \log \Gamma(N_k + \alpha) && (3.10)
\end{aligned}$$

A likelihood model for inference from positive observations

In order to define a model of probability updating, we need to define the likelihood of observing data set $\mathcal{D}$. We define a data set to be an ordered set $D = (O_1, \ldots, O_N)$ of *observations*, where each observation $O = (F_O, \mathcal{L}_O(\cdot))$ consists of collection of ground facts $F_O = f_1, \ldots, f_{|O|}$ and a likelihood function $\mathcal{L}_O(\cdot) : K \to [0, 1]$ from a knowledge base to the probability of observing F_O given that knowledge base.

Although there are many different and complex hypotheses that a learner might hold about how the data they observe is generated, in this work we will simplify matters by assuming that the probability of observing F_O is the probability of making the i.i.d observations of its elements, and we will assume that observations are picked uniformly at random without replacement from the set of all true *observable* facts Λ_K entailed by the knowledge base K.

Taken together, these assumptions imply that the probability of observing an observation O generated by a knowledge base K is:

$$\mathcal{L}(K|\mathcal{D}) = \prod_{O \in \mathcal{D}} \mathcal{L}(K|O) \tag{3.11}$$

$$\mathcal{L}(K|O) = \mathbf{1}(K \vDash F_O) \frac{(|\Lambda_K| - |F_O|)!}{|\Lambda_K|!} \text{ where } \Lambda_K \triangleq \{f | K \vDash f\}. \tag{3.12}$$

This likelihood function corresponds to the *size principle* for scoring hypotheses (see Tenenbaum 1999), which, though simple, has been shown to explain various phenomena in human concept generalization.

Example 3. *A paradigmatic case of a relational conceptual system is a kinship system. Consider a highly simplified kinship system that only consists of two observable binary relations* $\textsc{m}(X, Y)$ *(meaning X is a married to Y) and* $\textsc{c}(X, Y)$ *(meaning X is the child of Y), and defines* $\textsc{m}/2$ *and* $\textsc{c}/2$ *according to the clauses in 3-1.*

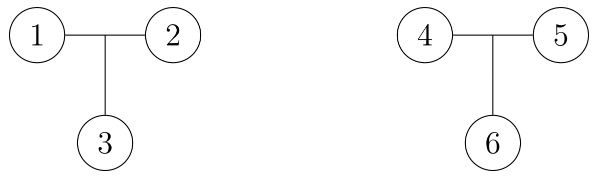
$$\textsc{m}(X, Y) \leftarrow \textsc{m}(Y, X) \tag{3.13}$$

$$\textsc{c}(X, Y) \leftarrow \textsc{c}(X, Z), \textsc{m}(Z, Y) \tag{3.14}$$

Figure 3-1: Clauses for a simplified kinship system containing the two binary relations $\textsc{m}(X, Y)$ (X is married to Y) and $\textsc{c}(X, Y)$ (X is the child of Y).

A complete specification of the relations exhibited by some set of individuals can be captured by writing down only a subset of the true relations among them; Figure 3-2 shows an example of this specification for two families consisting of individuals labeled $1, \ldots, 6$*. Pedigree graphs capture exactly this parsimony; each pedigree consists of two edges whereas an explicit specification of all true relations requires four facts per family. The relations are shown as a pedigree tree (ignoring gender)*

We can parameterize the learning task by the number of families represented in it. In this example, we will assume that each family consists of three people (two parents



$\textsc{m}(1,2)$	$\textsc{m}(4,5)$
$\textsc{m}(2,1)$	$\textsc{m}(5,4)$
$\textsc{c}(3,1)$	$\textsc{c}(6,4)$
$\textsc{c}(3,2)$	$\textsc{c}(6,5)$

Figure 3-2: The ground truth relations for Example 3

and a child) and that individuals are labeled by integers. Specifically, if we have N_f families in a data set, then there will be $3N_f$ possible individuals $\{0, \ldots, 3N_f - 1\}$. Let D_{N_f} be a data set consisting the complete set of entailed facts over N_f families:

$$\begin{aligned} D_{N_f} = &\{\textsc{m}(3n, 3n+1) | n \in \{0, \ldots, N_f\}\} \\ &\cup \{\textsc{m}(3n+1, 3n) | n \in \{0, \ldots, N_f\}\} \\ &\cup \{\textsc{c}(3n+3, 3n) | n \in \{0, \ldots, N_f\}\} \\ &\cup \{\textsc{c}(3n+3, 3n+1) | n \in \{0, \ldots, N_f\}\} \end{aligned} \tag{3.15}$$

To provide a feel for the hypothesis space that our model will be exploring, we present a possible learning scenario for this data, and a few points in the space of hypothetical knowledge bases. We define the knowledge base schema as

$$\Sigma = \left\{ \begin{array}{l} \exists PQ.P(X,Y) \leftarrow Q(X,Y) \\ \exists PQ.P(X,Y) \leftarrow Q(Y,X) \\ \exists PQR.P(X,Y) \leftarrow Q(X,Z), R(Z,Y) \\ \exists PQR.P(X,Y) \leftarrow Q(X), R(Y) \\ \exists PXY.P(X,Y). \\ \exists PX.P(X). \end{array} \right\} \tag{3.16}$$

Figure 3-3 shows three possible knowledge bases that entail data set D_{N_f}. K_{OVERFIT} in effect memorizes the data, defining each observed predicate as the disjunction of its observed instances. K_{UNDERFIT} defines both observed predicates as admitting any pair of individuals; this entails the data but also fails to capture the structure that will enable generalization. The third knowledge base $K_{\text{JUST RIGHT}}$ captures exactly the target theory. It is constructed by transforming the rules of the target theory into a form compatible with the knowledge base schema.

K_{UNDERFIT}	K_{OVERFIT}	$K_{\text{JUST RIGHT}}$
$\text{M}(X,Y) \leftarrow \text{C}(Y,X)$	D_{N_f}	$\text{M}(X,Y) \leftarrow \text{M}(Y,X)$
$\text{C}(X,Y) \leftarrow \ell_n(X), \ell_n(Y)$		$\text{C}(X,Y) \leftarrow \text{C}(X,Z), \text{M}(Z,Y)$
$\|n \in 1,\ldots,N_f$		$\text{M}(3n,3n+1), \text{C}(3n+2,3n)$
$\ell_n(i)$		$\|\, n \in 1,\ldots,N_f$
$\|n \in 1,\ldots,N_f$		
$i \in 3n, 3n+1, 3n+2$		

Figure 3-3: Three knowledge bases that entail D_{N_f} (see Figure 3.15). N_f is the number of families in the data set. K_{UNDERFIT} entails that the married-to and child-of relations hold between any two individuals in a family. To capture the unobserved same-family relation, this knowledge base uses a unary latent predicate for each family in the data set. K_{OVERFIT} entails all and only those facts found in the data set by recording each as a separate clause in the knowledge base. $K_{\text{JUST\_RIGHT}}$ entails the same clauses as K_{OVERFIT}, but directly records in the knowledge base only two of the four facts that hold in each family; the remainder are accounted for by its first two clauses.

3.5 MAP Inference for the IKM

In this section, we present HS-MAP-IKM, an algorithm for MAP inference in the IKM using heuristic state-space search. The algorithm uses the tools and techniques of classical state-space search in AI; all standard notation and terminology not covered here can be found in an introductory AI text such as Russell and Norvig (2010, chapter 3).

We consider the *maximum a posteriori* (MAP) inference problem for the IKM: given a data set D, the task is to find the knowledge base K^* that maximizes the probability of $p(K|D, \Sigma, \Theta)$. Broadly speaking, we can take either a *top-down* or *bottom-up* approach to this search problem. In the "top-down" approach, the algorithm generates hypothetical knowledge bases that are then scored against their fit to the data. In the "bottom-up" approach, which is by far the more popular in ILP algorithms, hypothetical knowledge bases are generated in some fashion directly from the data. The popularity of the bottom-up approach is due to the difficulty of finding, without direct appeal to the data, a knowledge base that entails the data. A bottom-up approach will usually take as the initial knowledge base just the data itself, which trivially entails itself. By framing the knowledge base induction problem as a heuristic state-space search, we combine the attractive features of the top-down and bottom-up approaches.

A key feature of the HS-MAP-IKM algorithm is that nodes in the state space correspond to pairs of partial schema instantiations (corresponding to partial knowledge bases) and associated partial proofs of the data set. As described Section 3.5, the knowledge base is constructed as a biproduct of a search over proofs of the facts in the data set. Thus, learning here is a form of abductive inference, where what is abduced is the rules in knowledge base that make that proof possible.

The state space The state space $\mathcal{S}$ is a directed graph. Each node $n = (G, \Delta) \in \mathcal{S}$ is associated with a *goal set* n.goals $= G$ and a schema instantiation $n.\Delta = \Delta$. Algorithm 5 defines the function CHILD which generates the children of nodes in $\mathcal{S}$. This function is defined by first selecting some goal node from the node's goal set (we discuss goal selection in Section 3.5) and then applying one of the two search operators GROUNDMETAVAR and BACKWARDCHAIN. The first of these has priority: that is, if the current node's selected goal contains metavariables, then GROUNDMETAVAR is used to nondeterministically instantiate one of the goal's metavariables with an element from its domain. The BACKWARDCHAIN operator, produces a child by either applying a standard backward-chaining step to the selected goal node using one of the clauses already in the node's knowledge base or by resolving the selected goal against the head of a metarule in Σ. In the latter case, a fresh metarule instantiation is inserted into the node's schema instantiation.

Algorithm 5: Generating the children of nodes in the IKM state space S.

```
  Def CHILD(node):
1     selected ← select(node.goals)
      if selected contains meta-variables then
          yield node in GROUNDMETAVAR(selected, node)
      else
          yield node in BACKWARDCHAIN(selected, node)
  Def GROUNDMETAVAR(selected, node):
      $V_1, \ldots, V_k$ ← metavars(selected)
      for $s_1 \in \mathbf{S}^{(\mathrm{ar}(V_1))}), \ldots, \mathbf{S}^{(\mathrm{ar}(V_k))}$ do
          $\sigma \leftarrow \{V_1 \mapsto s_1, \ldots, V_k \mapsto s_k\}$
          yield $\sigma(node)$
      end
  Def BACKWARDCHAIN(selected, node):
      $K \leftarrow K(\mathrm{node}.\Delta)$
      for $c \in K$ do
          $\sigma \leftarrow \mathrm{mgu}(selected$ :- $B, c)$
          node.goals ← remove(selected, node.goals)
          node.goals ← push($B$, node.goals)
          yield $\sigma$(node)
      end
      for metarule $r = \exists \bar{V}.H$ :- $B \in \Sigma$ do
          $\sigma \leftarrow$ mgu(selected :- $B, H$ :- $B$)
          node.$\Delta$ ← node.$\Delta \cup \{(r, \sigma$ restricted to $\bar{V})\}$
          node.goals ← remove(selected, node.goals)
          node.goals ← push($B$, node.goals)
          yield $\sigma$(node)
      end
```

Target nodes The goal of the state-space minimization problem is to find the minimal cost *target* node that is reachable from a given initial node. We define a node n to be a target node if its knowledge base is first-order and entails the observed data set. That is, n is a target node iff $K(n.\Delta)$ contains no higher-order variables, and $K(n.\Delta) \vDash D$.

We argue that given the search operators defined in Algorithm 5, if for some goal n, n.goals is empty and $n.\Delta$ is a complete schema instantiation, then n is a target node. The reasoning is as follows. Any higher-order variable in a knowledge base schema, must be a metavariable. Thus, if $n.\Delta$ is complete, and thus has no metavariables, then $K(\Delta)$ must be first-order. Further, consider a path from an initial node to a node with an empty goal set. The subset of arcs along this path that correspond to the search operator BackwardChain represent a backward chaining proof of the initial node's goal set, and by construction the clauses used in that proof are present in $K(n.\Delta)$.

Goal selection At line 1 of Algorithm 5, we need to specify how a goal should be selected from the current goal set. In the experiments here, the goal set is a LIFO queue: we always select from the goal set the goal that was most recently inserted. There may be other, more efficient, approaches. We do not explore these here, but we note that there are many possibilities, and that these roughly correspond to the dynamic variable ordering heuristics one finds in the constraint processing literature (see, e.g., Gent et al. 1996; Dechter 2003). One possibility is to select the goal with the smallest number of children; this is relatively cheap to assess as it does not involve computing the heuristic value for each child of each goal. Another possibility is to select the goal arbitrarily, thus introducing some stochasticity into the search procedure. The advantage of the LIFO approach is that the selection itself is very cheap. It also corresponds to a kind of intuitive notion of attention: while the heuristic function evaluates the state globally, the search moves themselves are confined the to the current subproblem.

A cost function Our state space minimization algorithm will be guided by a cost function $g(n)$ that assigns a cost to every node n in the state space graph. We consider first the function $g^* : S \mapsto \mathbf{R}$ which assigns a cost to every *target* node. It is simplest to frame the MAP inference problem as a minimization of the negative log probability: for any target node n, we set $g^*(n)$ to the negative logarithm of the joint probability of $K(n.\Delta)$ and $\mathcal{D}$ under our generative model. Making explicit g^*'s dependency on observed data set D, we have:

$$\text{if } n \text{ is a target node:} \tag{3.17}$$

$$g^*(n; \mathcal{D}, \Sigma, \Theta) \triangleq -\log p(\mathcal{D}, K | \Sigma, \Theta) \tag{3.18}$$

$$= -\log \mathcal{L}(K|\mathcal{D}) - \log p_{\text{prior}}(K \mid \Sigma, \Theta) \tag{3.19}$$

$$\text{where } K = K(n.\Delta) \tag{3.20}$$

We define cost function g to be an extension of g^* to non-target nodes, and we desire that this extension satisfy two basic properties:

- consistency: if a n is a target node, then $g(n) = g^*(n)$, and
- monotonicity: if n' is a child node of n, then $g(n') \geq g(n)$.

One option for defining g, which satisfies these two conditions, is simply to apply Equation 3.17 to all nodes in S; however, by construction, non-target nodes do not entail the data set, so $g^*(n)$ is ∞ for any non-target node. Thus, this option would result in a completely uninformative search space in which nearly all neighboring nodes have the same cost value.

In order to get a cost function that is more usefully graded, we compute the likelihood of a non-target node using a knowledge base augmented with that node's goal set. One sublety here is that both a non-target node's incomplete schema instantiation and its goal set may contain metavariables, and our likelihood function is only defined over first-order knowledge bases. At the level of the knowledge base, metavariables are existentially quantified logic variables that are not within the scope of any universally quantified logic variables. Therefore, we can *skolemize* existential variables by introducing a *Skolem constant* for each existential variable appearing in the knowledge base (Gabbay, Hogger, and Robinson 1998). In doing so, we introduce into the extension of observable predicates facts that contain Skolem constants; since our goal here is to find a lowerbound on the extension of observable facts, we do not count facts that contain Skolem constants towards the number of observable facts entailed by a partial knowledge base.

Example 4. *As a simple example of how this skolemization works, consider a data set consisting of the single goal $a(1)$ and a knowledge base schema that contains the metarule $\exists ABC.A(X) \leftarrow B(Y), C(X,Y)$. Suppose we perform a single* Backward-Chaining *step with this goal resolving against this metarule. This produces an incomplete knowledge base and a goal set that, when combined, give the following knowledge base:*

$$K = \exists BCZ. \begin{bmatrix} \forall XY.\, a(X) \leftarrow B(Y), C(X,Y) \\ B(Z) \\ C(1,Z). \end{bmatrix} \tag{3.21}$$

We skolemize variables A, B and C with constants f_A, f_B and f_C:

$$\begin{array}{rl} & \forall XY.\, a(X) \leftarrow f_B(Y), f_C(X,Y) \\ \textit{skolemize}(K) = & f_B(f_Z) \\ & f_C(1, f_Z). \end{array} \tag{3.22}$$

We only apply the skolemization step to the likelihood part of the cost function. This is because we do not know how these skolem constants are realized in the optimal extension of the partial knowledge base, and the prior probability of the unskolemized knowledge base is an upperbound on the that of its skolemized counterpart.

Finally, then, we arrive at the following cost function:

$$g(n; \mathcal{D}, \Sigma, \Theta) = -\log \mathcal{L}(K' | \mathcal{D}) - \log p_{\text{prior}}(K \mid \Sigma, \Theta) \tag{3.23}$$
$$\text{where } K = K(n.\Delta) \tag{3.24}$$
$$K' = \text{skolemize}(K \cup n.\text{goals}) \tag{3.25}$$

The intuition behind this cost function is that, under the prior, a node's knowledge base is no more costly than that of any of its descendants, and any descendant that is also a target will have a knowledge base that entails G. Therefore, both terms in the right hand side of Equation 3.23 are non-decreasing as one moves from a parent node to one of its children. This means that $g(\cdot)$ is monotonic. It is consistent because n.goals for any target node is empty, which implies that $g(\cdot)$ is equal to $g^*(\cdot)$ at target nodes.

Heuristic search for MAP inference *Heuristic search* is popular algorithmic framework in AI for systematically searching extremely large discrete spaces (Edelkamp and Schrödl 2012; Pearl 1984) within the *state-space search* paradigm. A heuristic search algorithm performs graph search with the aid of a *heuristic function* which provides an estimate of the distance from any node to the closest solution node. Although heuristic search algorithms do not in general improve the worst-case complexity of search problems in AI (the vast majority of which are NP-complete), they often improve the practical memory and time requirements of these problems by many orders of magnitude.

We use the following conventional notation for describing the heuristic search framework: as defined earlier, $g(n)$ is the cost of the current best path from n_0 to n. The heuristic function $h(n)$ is an estimate of the cost of the best path from n to a solution node. $f(n)$ is an estimate of the cost of the best path from n_0 to n^* that goes through n. In most cases, $f(n)$ is defined as $f(n) = g(n) + h(n)$, which is interpreted as saying that the cost of the best path that goes through n is the actual cost of the best path to n from the start node plus the estimate of the cost of getting to a goal node from n.

The performance of heuristic search algorithm depends greatly on the choice of heuristic function $h(\cdot)$. If the heuristic is inaccurate, then the algorithm can be misled into exploring parts of the search space in which no solution or in which suboptimal solutions exist. If the heuristic is perfect, then a greedy search will find an optimal solution. If a heuristic is *admissible* then it is guaranteed that some algorithms (like the very popular A^* algorithm; see Hart, Nilsson, and Raphael 1968), will find an optimal solution to the search problem.

Although constructing a good heuristic function for a given problem has traditionally been left to a human with some insight into the structure of the problem, research into efficient *domain general* planners has led to the development of techniques for automatically generating heuristics (Geffner and Bonet 2013). The basic insight is that if a problem can be framed as recursively solving multiple *dependent* subproblems, then a simpler problem can be automatically produced by systematically removing those dependencies. The simpler problem can then be used as an approximation for the original problem, and the cost of a solution in the simpler problem can be taken as a heuristic function for the cost of the solution in original problem.

In the problem considered here, the dependent subproblems correspond to the multiple goals in a node's goal set. The actions that incur a cost in our state-space are those in which the meta-interpreter needs to invent a rule or instantiate a metavariable in order to *explain* the current node. To explain a node, the meta-interpreter must explain each of the goals in its goal set. The best way to explain any one of these goals depends on how the others are explained, but according to the above technique, we can get a heuristic function by allowing the meta-intepreter to pretend that those costs are in fact independent. In doing so, we get the following heuristic $h(\cdot)$ (where we have left implicit the dependence on dataset D):

$$h((G,\Delta)) = \sum_{t\in G} h'(t,\Delta) \tag{3.26}$$

$$h'(t,\Delta) = \begin{cases} 0, & \text{if } t = \textsc{true}, \\ \min_{(G',\Delta')\in C_h((\{t\},\Delta))} (g((G',\Delta')) - g((\{t\},\Delta))) + h((G',\Delta)), & \text{otherwise}, \end{cases} \tag{3.27}$$

where C_h is a function that maps nodes to their children in the heuristic graph.

There are two sources of approximation in this heuristic. The first source is that in Equation 3.26, the heuristic value of the goal list is the sum of the heuristic values of each of the goals; this is an approximation because proofs of different goals in the same goal set are dependent on one another as they may bind variables commmon to both terms. The second source of approximation is that in Equation 3.27, the schema instantiation in the recursion term always remains Δ, not Δ'. This means that the heuristic cost of the child is computed with respect to the schema instantiation of the parent.

With these two approximations, $h'(\cdot)$ effectively only depends on a single goal term. The "true" cost function $g(\cdot)$, on the other hand, depends on a list of goal terms and a schema instantion. This difference between $h'(\cdot)$ and $g(\cdot)$ is what makes computing the heuristic efficient compared to computing the original cost function.

We can frame the computation of $h((G,\Delta))$ as minimization in a *relaxed state-space* $\mathcal{S}_{\text{rel}}(G,\Delta)$ parameterized by the node (G,Δ) whose heuristic valued is being determined. Unlike the original state space $\mathcal{S}$, $\mathcal{S}_{\text{rel}}$ is an AND/OR graph, in which AND nodes sum over the costs of their independent children, and OR nodes perform the non-deterministic search operations of the original search space. This state-space is fully specified by a cost function over the edges of the $\mathcal{S}_{\text{rel}}$ and the two search operators $\mathcal{C}^{\text{OR}}_{\text{rel}}$ and $\mathcal{C}^{\text{AND}}_{\text{rel}}$ which map OR and AND nodes, respectively, to their children (see Algorithm 6).

Note that lines 1 and 2 correspond to two different choices in generating AND nodes. In the first, an OR node has a single child, namely, the AND node that is labeled with the goals resulting from the search operator chosen at the parent. In the second, there is an additional step: if the result of the parent search operator has multiple goals that share variables among them, then those variables are first instantiated non-deterministically, thus breaking the variable-induced dependency between those goals and making the heuristic relaxation more accurate. There is a trade-off, of course, as instantiating these shared variables can dramatically increase the number of distinct nodes in $\mathcal{S}_{\text{rel}}$. In the results we describe in this paper, we always use line 2. Example 5 describes the effect of instantiating shared variables and motivates our choice.

```
Algorithm 6: Search operators in $\mathcal{S}_{\text{rel}}(G_0, \Delta_0)$
  $G$: a goal set
  $\Delta$: a knowledge base schema instantiation
  $C_{\mathcal{S}}$: the search operators of original state-space $\mathcal{S}$
  Def $\mathcal{C}^{\text{OR}}_{rel}(G, \Delta)$:
      if $|G| = 1$ then
          return $\{\text{OR}(G', \Delta') | (G', \Delta') \in C_{\mathcal{S}}((G', \Delta_0))\}$
      else
1           (*) return $\text{AND}(G, \Delta)$
2           (**) return $\{\text{AND}(G', \Delta') | (G', \Delta') \in \textsc{InstSharedVars}(G', \Delta_0)\}$
  Def $\mathcal{C}^{\text{AND}}_{rel}(G, \Delta)$:
      return $\{\text{OR}(\{t\}, \Delta') | t \in G\}$
  Def $\textsc{InstSharedVars}(G, \Delta)$:
      $Vs \leftarrow$ shared variables in $G$
      return $\{(G', \Delta') | G' \textit{ is } G \textit{ with vars in } Vs \textit{ instantiated }\}$
  Def $\textsc{ArcCost}((G_p, K_p), (G_c, K_c))$:
      return $g((G_c, K_c)) - g((G_p, K_p))$
```

Example 5. *Suppose that we have a knowledge base*

$$K = \left\{ \begin{array}{l} C_1 = a(X,Y) \leftarrow a(X,Z), b(Z,Y) \\ C_2 = a(1,1) \\ C_3 = b(2,2) \end{array} \right\} \tag{3.28}$$

and a schema

$$\Sigma = \left\{ \begin{array}{l} M_1 = \exists AXY.A(X,Y) \\ \dots \end{array} \right\}. \tag{3.29}$$

Then there at least two ways to prove the goal set $G = \{a(1,2)\}$ in $\mathcal{S}$ (we display the proofs as a sequence of goal sets, where the underlined goal is the selected goal, and each arrow is a backward chaining step marked with the used clause or metarule and substitution):

$$\underline{a(1,2)} \xrightarrow{C_1:(X,Y)\mapsto(1,2)} \underline{a(1,Z)}, b(Z,2) \tag{3.30}$$

$$\xrightarrow{C_2:Z\mapsto 1} \underline{b(1,2)} \tag{3.31}$$

$$\xrightarrow{M_1:(A,X,Y)\mapsto(b/2,1,2)} \text{TRUE} \tag{3.32}$$

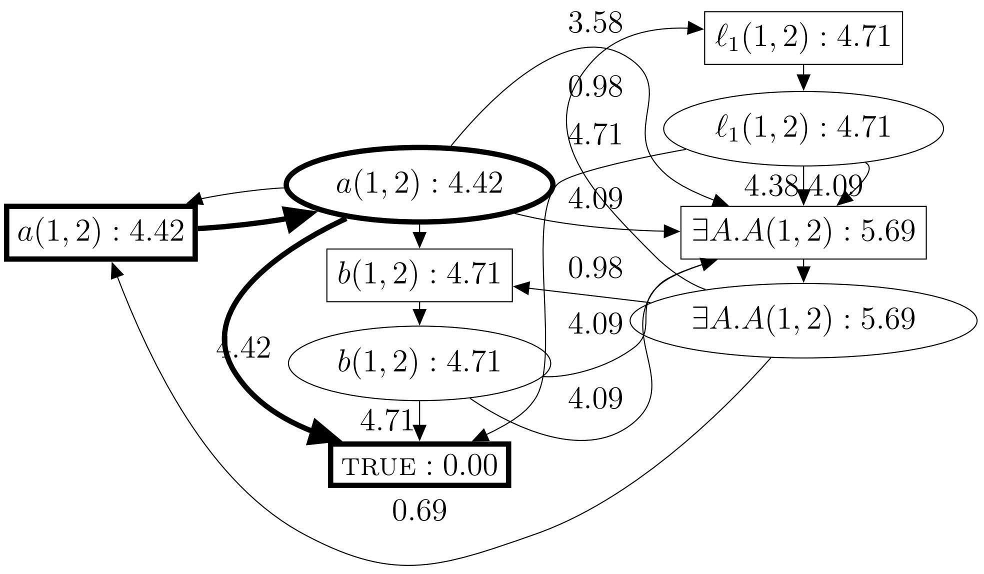
and

$$\underline{a(1,2)} \xrightarrow{M_1:(A,X,Y)\mapsto(a/2,1,2)} \text{TRUE}. \tag{3.33}$$

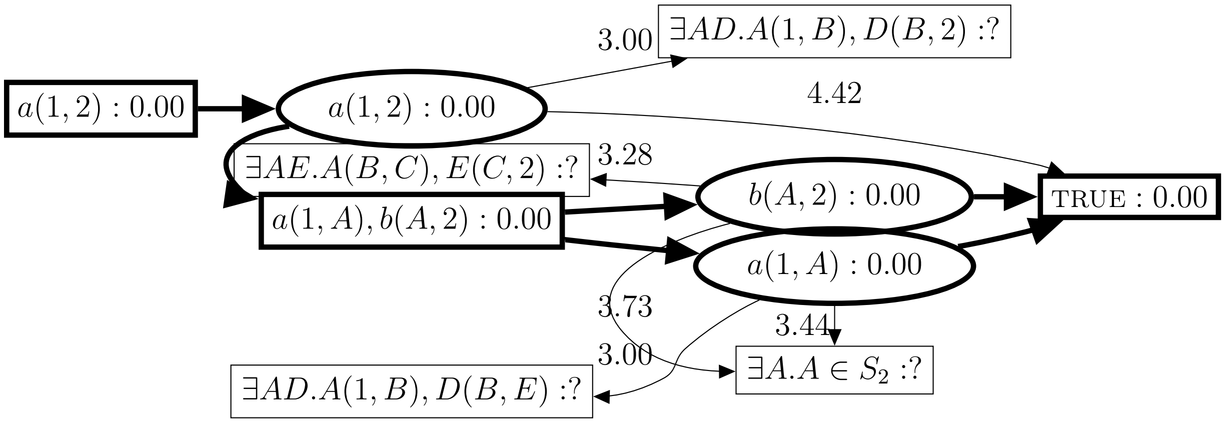
Each of these two proofs incurs the cost of adding a binary fact to the knowledge base, and which is the better proof depends on which of the two predicates $a/2$ and $b/2$ has a higher prior probability. If the knowledge base above represented some intermediate node in our MAP search procedure, then we would expect that the second proof have the lesser cost because K has three occurances of predicate symbol $a/2$ but only two of predicate symbol $b/2$. This is a consequence of the "rich-gets-richer" property induced by our use of a CRP over symbol instantiations.

Now we can compare these proofs to our heuristic. Figure 3-4b shows $\mathcal{S}_{rel}$, using line 1 of the Algorithm 6. The heuristic cost for proving $a(1,2)$ is 0. This is because the goals $a(1,X), b(X,2)$ which cannot be satisfied in K are effectively "relaxed" to the goals $a(1,X), b(Y,2)$, which can. Therefore, in the heuristic graph, there exists a relaxed proof of $a(1,2)$ in which no additional metarule instantiations are required. By contrast, Figure 3-4a shows the heuristic graph when we use line 2 of Algorithm 6. In this case, the dependency between the previously mentioned goals is maintained, resulting the more accurate heuristic estimate in which adding $a(1,2)$ is less costly that $b(1,2)$.

Putting it together Given the ingredients define above, any of a number of heuristic search algorithms can be used. In the case studies that follow we use the basic A^* algorithm. We leave a more thorough investigation of different heuristic search algorithms for future work.



(a) Instantiating shared variables.



(b) Without instantiating shared variables.

Figure 3-4: The effect of instantiating shared variables on the explored heuristic graph $\mathcal{S}_H$. Rectangular nodes are OR nodes; oval nodes are AND nodes. $\mathcal{S}_H$ is expanded using depth-first branch and bound. Where a node's arcs have been pruned by the branch and bound procedure, the node cost is left as a question mark; otherwise, the node is labeled with its optimal cost.

3.6 Experiments and case studies

In this section, we explore the empirical behavior of the HS-MAP-IKM algorithm.

3.6.1 Dealing with noisy data

One of the limitations of classical approaches to symbolic concept learning is the difficulty of dealing with noisy data. Bayesian approaches to concept learning naturally accodomate noisy data. This can be done in one of two ways: either a noise generating process is assumed to exist on top of the underlying "pure" concept definition or noisy observations are included within the concept definitions themselves.

We take the latter approach: noisy observations are accomodated directly within the learned conceptual representation. If an exceptional fact is found in the data, it can be incorporated into the underlying knowledge base as a fact; and, importantly, if there is some structure to the noise generating process, then that structure can be incorporated as rules in the knowledge base.

To explore how our model behaves in the presence of noisy data, we ran an experiment based on the task shown in Example 3. We compare the log probability of a knowledge base learned using the HS-MAP-IKM to the manually created "landmark" knowledge bases (K_{UNDERFIT}, K_{OVERFIT}, and $K_{\text{JUST RIGHT}}$) as we increase both the problem size and level of noise.

Methods The scenario and schema definitions in this experiment are the same as those in Example 3.

We generated 10 sample data sets for each noise level $\mu \in 0, 0.2, 0.4$ and for each $N_f \in 1, 3, \ldots, 11$. To generate a noisy data set $D_{N_f}[\mu]$ of N_f families with with noise level $0 < \mu < 1$, we take the noiseless data set D_{N_f} and, for each relation between two individuals in the same family, we toggle its presence or absence in the data set with probability μ. That is,

$$D_{N_f}[\mu] = D_{N_f} \oplus \{P(X,Y) | P \in \{\text{M}, \text{C}\}, X, Y \text{ same family }, \text{ with prob. } \mu\}. \quad (3.34)$$

While K_{UNDERFIT} entails the noisy data sets without modification, we extend the other two knowledge bases for a noisy data set $D_{N_f}[\mu]$ as follows:

- $K_{\text{OVERFIT}}(D_{N_f}[\mu]) = D_{N_f}$
- $K_{\text{JUST\_RIGHT}}(D_{N_f}[\mu]) = K_{\text{JUST\_RIGHT}} \cup \{f | f \in D_{N_f}[\mu] \backslash D_{N_f}\}$

That is, $K_{\text{JUST\_RIGHT}}(D_{N_f}[\mu])$ just memorizes any observed facts that it cannot otherwise explain.

To compute K_{LEARNED}, we used the incremental version of HS-MAP-IKM with a time limit of 2 minutes and with each batch of observations consisting of two families. We set $\alpha = 0.5$, and $\delta = 0.75$.

For the analysis shown in Figure 3-5, we compute on the y-axis the log probability of the given theory relative to that of K_{OVERFIT} and normalized the number of families.

That is, the bar associated with knowledge base K at noise level μ and number of families N_f:

$$\frac{1}{N_f}(\log p(K;\mu,N_f) - \log p(K_{\text{OVERFIT}};\mu,N_f)). \tag{3.35}$$

Error bars show standard error.

Results When no noise is present $K_{\text{JUST RIGHT}}$ is consistently the best theory, as expected, and K_{LEARNED} is nearly as good. For the low level of noise $\mu = 0.2$, the performance of the K_{LEARNED} suffers somewhat relative to $K_{\text{JUST RIGHT}}$. This is likely because the simple noisy model assumed for $K_{\text{JUST RIGHT}}$ does well enough when the low levels are low, and the noisy data makes the search over knowledge bases more challenging. However, as the noise increases to $\mu = 0.4$, $K_{\text{JUST RIGHT}}$ becomes clearly inadequate, especially when the amount of data is small. For the smallest amount of data, K_{LEARNED} does just as well as K_{OVERFIT}, and as the amount of data increases, it begins to discover the structure in the data while consistently doing better than $K_{\text{JUST RIGHT}}$ and K_{UNDERFIT}.

3.6.2 Does the heuristic improve learning times?

A basic empirical question is whether the heuristic we have described is effective in the sense that using the heuristic improves search performance over not using it. This can be assessed by looking at the amount of effort learning requires with and without the aid of the heuristic. We evaluate this empirically by looking at how learning time and the number of search states evaluated during learning grow as a function of problem size.

Methods We set up two learning tasks, BIPARTITE$_n$and STAR$_n$, each of which is parameterized by problem size $n = 1, 2, \ldots$. The ground truth theories for BIPARTITE$_n$ and STAR$_n$ are shown in Figures 3-6b and 3-6c, respectively. We use the same the KB schema, displayed in Figure 3-6a, for both learning tasks. Because, in the absence of a heuristic, search is computationally challenging even for small schemas, the schema used here contains only three rules, the minimum needed to express the target theories.

For each n, the data set for each task consisted of all the the entailed by the corresponding theory. Learning was accomplished using a basic A^* algorithm, run until the first solution was returned, for each of two conditions: in the +HEURISTIC condition, the astar algorithm is

In Figure 3.6.4, we show both the number of nodes in the search space expanded and the time elapsed as we increase the size of the problem. In figure 3-6d, we show this comparison for two learning problems, versions of the BIPARTITE and STAR learning tasks (see Figures 3-6b and 3-6c, respectively). We provide the algorithm with a very simplified knowledge base schema (shown in Figure 3-6a), due to the limited computational tractability of search without a heuristic.

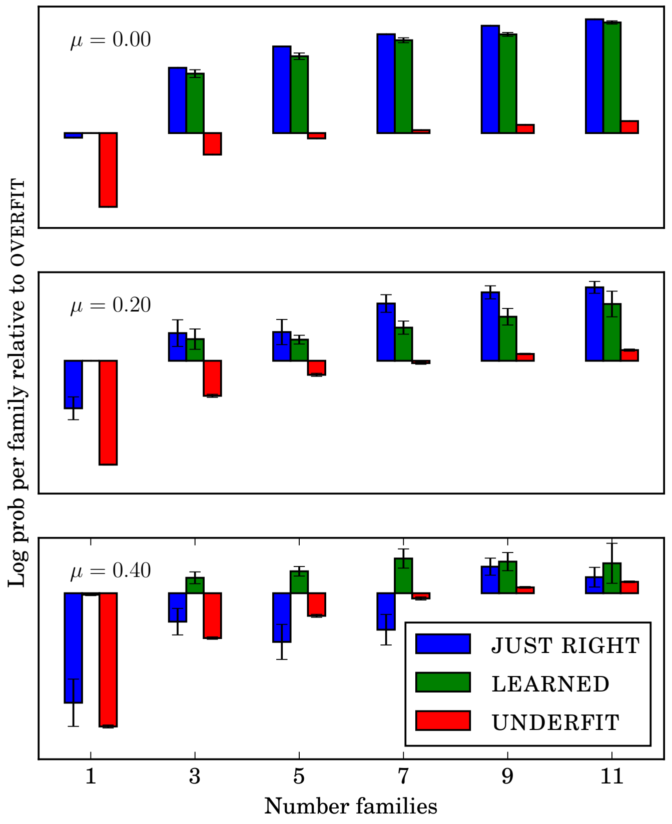


Figure 3-5: A comparison of HS-MAP-IKM against our other handwritten knowledge bases in the presence of noisy data. The task setup and the definitions of the knowledge bases is the same as found in Example 3. μ is the error probability, i.e., the probability that a given true fact is not in the dataset or that a false fact is. The height of each bar is the log probability of a given knowledge base per family member relative to what it would be under the overfitting knowledge base (which memorizes the data) K_{OVERFIT}.

In this experiment, as in most of the other ones in this paper, we employ a basic A^* graph-search algorithm. In -HEURISTIC condition, no heuristic function is used. In the +HEURISTIC condition, the HS-MAP-IKM$^+$ algorithm is used. We ran the search algorithms until the first goal was found.

Results As expected, the data show that in the -HEURISTIC both the number of nodes explored and the time taken to find this solution grow exponentially with the problem size. The size of the data set grows linearly for the STAR theory, whereas it grows quadratically for the BIPARTITE theory; this explains the much faster increase in the time and nodes expanded in the latter theory. Moreover, as expected, we find see that the search effort is much less in the +HEURISTIC condition, showing what

$$\begin{aligned} &\exists AB.A(X,Y) \leftarrow B(X,Y) && \text{with } S_0 = \{x_1,\ldots,x_n,y_1,\ldots,y_n,\ell_1^{(0)},\ell_2^{(0)},\ldots\} \\ &\exists ABC.A(X,Y) \leftarrow B(X),C(Y) && S_1 = \{x_1,\ldots,x_n,y_1,\ldots,y_n,\ell_1^{(1)},\ell_2^{(1)},\ldots\} \\ &\exists AX.A(X) && S_2 = \{a,\ell_1^{(2)},\ell_2^{(2)},\ldots\} \end{aligned}$$

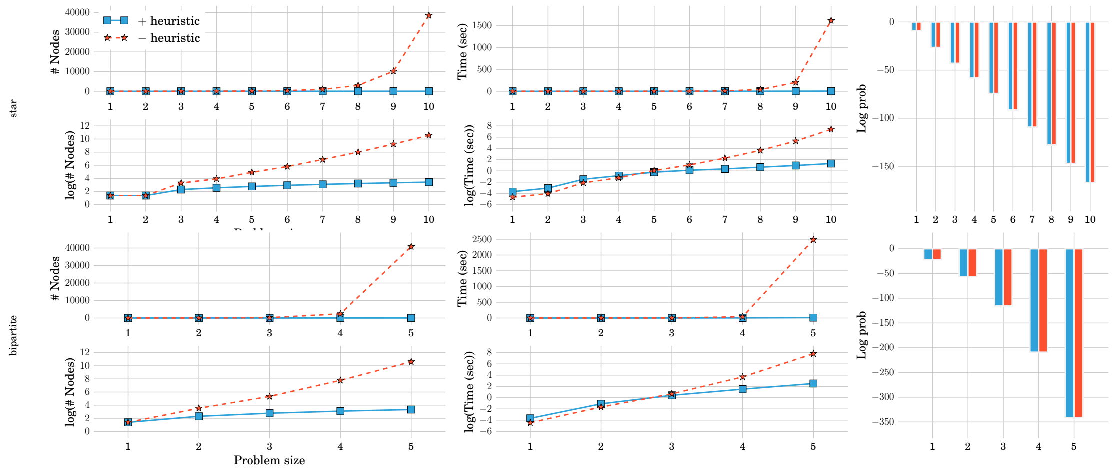
(a) KB Schema

$$\begin{aligned} &a(X,Y) \leftarrow \ell_1(X),\ell_2(Y) \\ &\ell_1(x_1),\ell_1(x_2),\ldots,\ell_1(x_n) \\ &\ell_2(y_1),\ell_2(y_2),\ldots,\ell_2(y_n) \end{aligned}$$

(b) BIPARTITE target theory

$$\begin{aligned} &a(X,Y) \leftarrow \ell_1(X),\ell_2(Y) \\ &\ell_1(x_1) \\ &\ell_2(y_1),\ell_2(y_2),\ldots,\ell_2(y_n) \end{aligned}$$

(c) STAR target theory



(d) Search effort as problem size grows.

Figure 3-6: Gauging the effectiveness of the HS-MAP-IKM heuristic function. This figure compares the search effort and solution quality with and without this heuristic. This comparison is accomplished by paramterizing two tasks (specified at the top of the figure) as a function of a problem size parameter n. The bottom of the figure compares the number of nodes expanded, the search time, and the log probability of the found solution between these the two versions of the search. For number of nodes expanded and search time, we also show the values on a log-scale. We see that even though searching with the heursitic scales much better than doing so without it, we obtain solutions of equivalent quality in all cases.

seems to be asymptoticallly faster search. Nor is it the case that the cost of computing the heuristic renders the +HEURISTIC the slower method in practical terms. (This last point is important because unlike the technique in much of the heuristic search literature, our heuristic cannot be precomputed, as its value at a given state depends the partial knowledge base at that state, and the number of these partial knowledge bases is unbounded.) Finally, we note that the speed of the +HEURISTIC condition does not come at the cost of optimality in this case; this is shown in the bar plots in Figure 3-6d which show the log probability of the goal knowledge bases for each of the conditions. The use of A^* means that the optimal solution is found in the -HEURISTIC case; but since our heuristic is not admissible, we have no gaurantee of finding an equivalently good solution in the +HEURISTIC condition. While we cannot make any claims about the approximate optimality of the algorithm in the general case, in these data at least, we are not improving the speed of the search procedure by finding lower quality solutions.

3.6.3 Heuristic graph size

A primary concern with our approach to computing the HS-MAP-IKM heuristic is that the computational cost of doing so does not scale well with the problem size. In this experiment we examine empirically the computational cost of computing this heuristic.

The computational cost of computing the heuristic is determined by the number of arcs traversed in the heuristic graph; this itself is determined by the number of nodes in the graph. Each OR node in the graph is labeled with exactly one goal, and each AND node in the graph is labeled with a conjunction of goals. Except for the root node, AND nodes can not be labeled with more D goals, where D is the size of the largest antecedent of in the knowledge base or knowledge base schema. Each goal in the conjunction is a positive literal in the Herbrand universe of the knowledge base. Let the size of the Herbrand universe be H. Then the number of nodes in the heuristic graph is $O(|H|^D)$. Each arc in the graph is between and AND node and an OR node. The number of OR nodes is $O(|H|)$. So the number of arcs is $O(|H|^{D+1})$.

Methods To empirically evaulate the computational cost of computing the heuristic, we evaluated the heuristic for the following state as a function of N and K, where N is the number of non-latent constants in the knowledge base schema and D is the size of the largest "chain" clause schema.

Let Σ_D be

$$\exists AB.A(X,Y) \leftarrow B(X,Y) \qquad \text{with } S_0 = \{x_1, \ldots, x_n, \ell_1^{(0)}, \ell_2^{(0)}, \ldots\} \tag{3.36}$$

$$\exists ABC.A(X,Y) \leftarrow B_1(X,Z_1), B_2(Z_1,Z_2), \ldots, B_D(Z_{D-1},Y) \qquad S_1 = \{\ell_1^{(1)}, \ell_2^{(1)}, \ldots\} \tag{3.37}$$

$$\exists AX.A(X) \qquad S_2 = \{a, \ell_1^{(2)}, \ell_2^{(2)}, \ldots\} \tag{3.38}$$

$$\exists AXY.A(X,Y) \tag{3.39}$$

and K_N be

$$\text{ANY}(X, Y) \leftarrow \text{ANY}(X), \text{ANY}(Y) \tag{3.40}$$
$$\text{ANY}(X) \leftarrow X \in 1, \ldots, N. \tag{3.41}$$

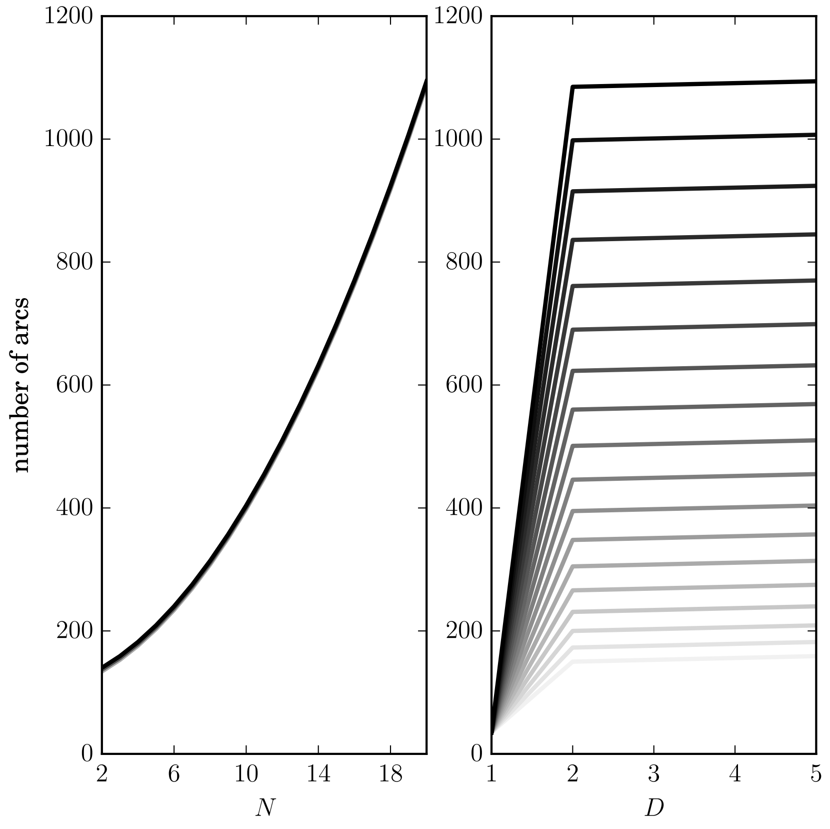


Figure 3-7: The number of arcs in the heuristic graph as a function of the number of constants N and the maximum schema clause degree D in the schema. The different lines in the right graph are the different levels of N; darker lines indicate larger N. On the left, the differences between different levels of D is impercetible.

Results Figure 3-7 shows how the number of arcs N_{arcs} in the heuristic graph grows as a function of the number of constants N_{const} and the maximum clause degree D of the schema. N_{arcs} seems to be insensitive to D beyond $D = 2$, suggested that the AND/OR branch-and-bound search we are using to explore the $\mathcal{S}_{\text{rel}}$ is pruning the use of clauses with degree greater than two. However, there seems to be no similar attenuation with respect to N_{const}.

The polynomial dependency of N_{arcs} on the number of symbols in the knowledge base is particularly concerning given that our model allows for an increasing number of symbols as learning progresses. For this heuristic to be viable, it will be important construct further approximations that limit this dependency.

3.6.4 Comparison with METAGOL

As mentioned in Section 3.3, METAGOL$_D$ performs an iterative deepening search on the size of the knowledge base induced in the meta-proof of the data. Since the iterative-deepening procedure, if continued indefinitely, will eventually discover every knowledge base that entails the data, we can use METAGOL$_D$ to optimize any objective function. We do this by maintaining an upperbound on the cost of the optimal knowledge base. We run the iterative deepening search for as long as time allows; whenever we find a target knowledge base, we update our upperbound, and whenever a partial knowledge base exceeds the current upperbound, we backtrack. This is in effect what METAGOL$_O$ does, with the modification that the "resource complexity" cost that METAGOL$_O$ uses is a function of the proof so far and not the knowledge base so far. That is, in METAGOL$_O$, if a particular first-order clause is used more than once, a cost is incurred each time. In the METAGOL variant described above, which we will call METAGOL$_{D(\text{IKM})}$, the IKM is associated with the partial knowledge base, not the partial proof.

Here, we compare HS-MAP-IKM with METAGOL$_{D(\text{IKM})}$. Because METAGOL$_{D(\text{IKM})}$ does not use the heuristic function to direct the search, it does not need to traverse the heuristic graph for each expanded arc in the $\mathcal{S}$, and therefore the rate at which it explores partial knowledge bases is much greater than that of HS-MAP-IKM. The empirical question is whether the benefit of using the heuristic function to guide the search compensates for the overhead of computing the heuristic function itself.

Methods To explore this question, we ran both algorithms on data from five relational theories. For each of these these five theories, the data set learned from is the set of entailed facts of predicate $a/2$. These target theories are shown in the left two columns of Table 3-9, and the knowledge base schema used by both algorithms is shown in Figure 3-8. The learners were supplied two background knowledge predicate, $obj/1$ and $eq/2$. $obj(X)$ is true when for $X \in 1, \ldots, 8$ for all theories except for TRANSITIVE, in which case $X \in 1, \ldots, 6$. $eq(X, Y)$ is true whenever $X = Y$. The knowledge base was paramterized with parameters $\alpha = 0.5$ and $\gamma = 0.75$, and each algorithm was run for ten minutes on each dataset.

Results The right two columns of Table 3-9 shows that best theory found by each algorithm. For SUBGROUP and STAR + EQ, both algorithms find theories that are syntactically identical to the target theories from which the data set was generated. These two theories correspond have the shortest target theories. In the time allotted, METAGOL$_{D(\text{IKM})}$ does not discover a knowledge base for the other three data sets that is better than the trivial definition of $a/2$ as the complete relation over pairs of objects. For all theories, HS-MAP-IKM finds knowledge bases with the target extension, though its knowledge base for SUBGROUP + EXCEPTION and TRANSITIVE

$$\begin{aligned}
&\exists Q, X, Y.\, Q(X,Y)\\
&\exists Q, X.\, Q(X)\\
&\exists Q, P.\, Q(X,Y) \leftarrow P(X,Y)\\
&\exists Q, P.\, Q(X,Y) \leftarrow P(Y,X)\\
&\exists Q, P.\, Q(X) \leftarrow P(X)\\
&\exists Q, P, R.\, Q(X,Y) \leftarrow P(X,Z), R(Z,Y)\\
&\exists Q, P, R.\, Q(X,Y) \leftarrow P(X,Y), R(X,Y)\\
&\exists Q, P, R.\, Q(X,Y) \leftarrow P(X), R(X,Y)\\
&\exists Q, P, R.\, Q(X,Y) \leftarrow P(X), R(Y,X)\\
&\exists Q, P, R.\, Q(X,Y) \leftarrow P(X), R(Y)
\end{aligned}$$

Figure 3-8: The metarules use in the comparison of the IKM to METAGOL.

are suboptimal with respect to the prior probability disribution on knowledge bases. The HS-MAP-IKM solution for SUBGROUP + EXCEPTION contains two copies of the identical clause; such syntactic redundancies could be easily detected and removed. In contrast, the suboptimality in the HS-MAP-IKM solution for TRANSITIVE is less simple.

The time courses of log probability for each theory is shown in Figure 3-10. Roughly, as the number of clauses in the target theory increases, we see HS-MAP-IKM performing better than METAGOL$_{D(\mathrm{IKM})}$. In the case of STAR + EQ, the target theory has only three clauses, and iterative-deepening search finds the target solution much faster than the heuristic guided search. For the SUBROUP theory, where the number of clauses in the target theory is increased by one, we see a mixed result, with HS-MAP-IKM finding a close to optimal solution faster than METAGOL$_{D(\mathrm{IKM})}$ even though it is somewhat slower to find the optimal one (here, we take the "optimal" solution to be the target solution, even though we do not prove that the target knowledge bases we have provided are in fact the globally optimal solutions). As stated above, for the remainder of the theories, whose target theories have between five and seven clauses, METAGOL$_{D(\mathrm{IKM})}$ fail to find a non-trivial solution.

To an extent, these results are to be expected given those shown in Figure . There are various differences between these results and the previous ones, but a significant one is that we use here the implementation of METAGOL$_{D(\mathrm{IKM})}$ provided by its authors (with minor modifications to maintain the negative log probability upper bound). In the previous results, we merely lesioned the heuristic computation from an implementation designed for its use. We think, therefore, that these results argue strongly that although computed the heuristic function is costly, the benefits of much faster node expansion is quickly outweighted by the exponentially increasing search space.

	target theory	HS-MAP-IKM	METAGOL$_{D(\mathrm{IKM})}$
subgroup	$a(X,Y) \leftarrow \ell_1(X), obj(Y)$ $\ell_1(6).\ell_1(7).\ell_1(8).$	$a(A,B) \leftarrow obj(A), \ell_2(B)$ $\ell_2(6).\ell_2(7).\ell_2(8).$	$a(A,B) \leftarrow obj(A), \ell_2(B)$ $\ell_2(6).\ell_2(7).\ell_2(8).$
subgroup + exception	$\ldots$subgroup theory$\ldots$ $a(1,1)$	$a(1,1).$ $\ell_2(8).\ell_2(7).\ell_2(6).$ $a(F,G) \leftarrow obj(F), \ell_2(G)$ $a(K,L) \leftarrow obj(K), \ell_2(L)$	$a(A,B) \leftarrow obj(A), obj(B)$
star + eq	$a(X,Y) \leftarrow \ell_1(X), obj(Y)$ $\ell_1(1)$ $a(X,Y) \leftarrow eq(X,Y)$	$a(A,B) \leftarrow eq(A,B)$ $a(E,F) \leftarrow \ell_2(E), obj(F)$ $\ell_2(1)$	$a(A,B) \leftarrow eq(A,B)$ $a(E,F) \leftarrow \ell_2(E), obj(F)$ $\ell_2(1)$
symmetric	$a(X,Y) \leftarrow a(Y,X)$ $a(3,4).a(1,2).a(1,3)$ $a(1,3).a(5,6).a(2,7).$	$a(2,7).a(5,6).a(1,3).$ $a(1,2).a(3,4).$ $a(E,D) \leftarrow a(D,E)$	$a(A,B) \leftarrow obj(A), obj(B)$
transitive	$a(X,Y) \leftarrow a(X,Z), a(Z,Y)$ $a(1,2).a(2,3).a(3,4).$ $a(4,5).a(5,6)$	$a(A,C) \leftarrow a(A,B), a(B,C)$ $a(5,6).a(4,5).a(4,5).$ $a(2,3).\ell_2(1).$ $a(L,M) \leftarrow \ell_2(L), obj(M)$	$a(A,B) \leftarrow obj(A), obj(B)$

Figure 3-9: Five relational theories on which we compare HS-MAP-IKM and METAGOL$_{D(\mathrm{IKM})}$. The first column is the ground truth theory, i.e., the the theory from which the dataset was generated. The second column is the best theory that our algorithm found. The third column is the best knowledge base that METAGOL found. METAGOL$_{D(\mathrm{IKM})}$ returns trivial underfitting theories in three of the five cases. HS-MAP-IKM finds solutions worse than ground truth in the two of the five cases, but it's solutions still manage to capture much of the statistical structure of the dataset in those cases. The corresponding search timecourses are shown in Figure 3-10.

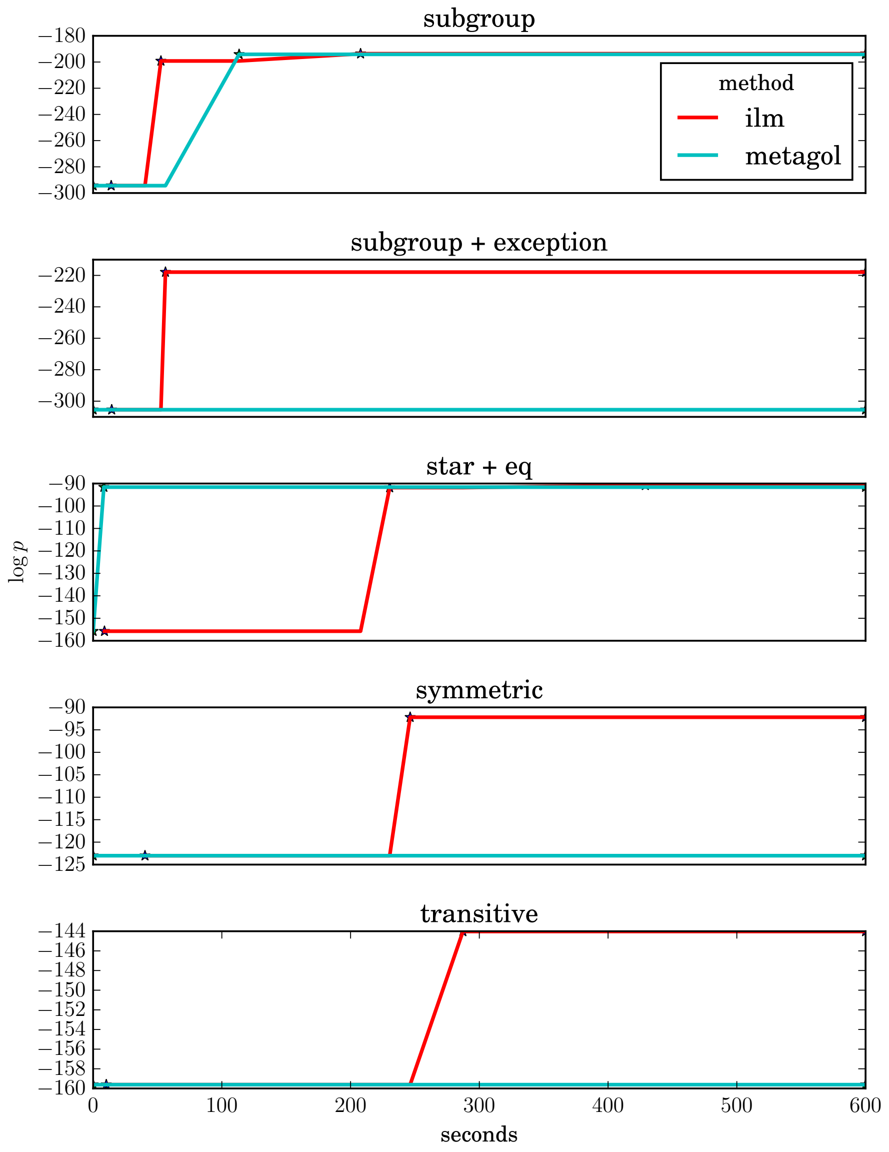


Figure 3-10: Comparing the search timecourses of HS-MAP-IKM and METAGOL$_{D(\mathrm{IKM})}$ on five relational theories. Ground truth for each task and the discovered theories for each algorithm are shown in the 3-9. Generally, we see that within the ten minute limit of this experiment, HS-MAP-IKM is able to more effectively escape local optima found early in the search.

3.7 Conclusion

Humans flexibly and reliably acquire knowledge about systems of interdependent relational concepts. The work presented here continues a long line of research that attempts to understand how an automated learning system can achieve the kind of efficiency, reliability and expressivity that even the youngest children seem to exhibit in their learning behaviors. The basic mystery at the root of this question is how learners deal with a hypothesis space of knowledge bases that is both enormously large and very non-smooth. We have argued here that by framing the learning problem as posterior inference in a hierarchical probabilistic model and then applying the techniques of heuristic search, we can get traction on this problem of practical tractability.

The algorithmic approach we have presented, however, relies on limiting our knowledge representation language in significant ways. We have restricted ourselves to negation-free datalog. Introducing negation in the body of clauses will introduce non-monotonicity to the model's likelihood function; as we learn new facts about one relation, we may be *reducing* the number of entailed facts for another relation. Adding negation would also introduce well-known issues in the semantics and evaluation of negation in datalog.

Though the heuristic we have introduced provides us with traction on the very difficult search problem, the heuristic itself is computationally costly: the size of the heuristic graph grows polynomially with the number of symbols in the knowledge base and that number is usually quite large in any knowledge base of useful size. Therefore, it will be important to find ways to limit the size of the heuristic graph or to develop proxies that provide a similar guiding signal in the search space.

The motivation for this work is the intuition that the solution to automated learning of conceptual knowledge will be found at the intersection of representational and algorithmic considerations. Much of the algorithmic work in AI has been focused on solving problems in discrete combinatorial spaces, where it is the job of the AI practitioner to model the domain of interest within these limited representations. We have attempted to show here that we should remove that practioner out of the loop while still using the same algorithmic techniques, and that we can do so by embedding a symbolic representation langauge within the paradigm of state-space graph search.

Chapter 4

Modeling intuitive theories with the IKM

4.1 Introduction

Our common sense knowledge about a particular concept (MOTHER, for example) goes well beyond knowing how to identify its instances. We know how concepts relate to one other (FATHER, CHILD, FAMILY) via the abstractions and rules that give them meaning (PARENT, SEX, RELATIVE). This interrelated system of common sense concepts is called an *intuitive theory*, and understanding how these intuitive theories are acquired, represented, and used is a central aim of cognitive science.

How do we account for the people's ability to learn these systems of rules and abstractions? How do they determine how many rules to use and how many abstract concepts to invent? How do they know which observed relations are true "just because" and which are explained by other more intrinsic relations? And, finally, how do they do so from small noisy observations of positive data.

We argue here that the *Infinite Knowledge Base Model* (IKM) provides such an account. This model is a particular instantiation of the Bayes Language of Thought (LOT) hypothesis – the hypothesis the flexibility and robustness of human conceptual learning is the result of learning mechanisms which effectively perform Bayesian inference over a hypothesis-space consisting of sentences in symbolic formal language. The IKM's particular instantiation of this hypothesis combines the expressive representational formalism of first-order deductive databases (a kind of restricted first-order logic) with the flexibility and robustness of nonparametric Bayesian statistics.

In this paper, we situate the IKM within the Bayes LOT literature so far, and demonstrate with a series of case studies that it goes beyond previous work in several concrete ways: it provides a generative model over knowledge bases with an unbounded number of invented predicates; it provides a learning approach that moves away from the intractable and psychologically improbably generate-and-test algorithms used by much of the literature; and it does so while still providing a general-purpose model, capable of expressing several qualitatively different kinds of concept learning.

Model	Representation	First order	Syntactic restriction	Predicate invention	Recursion	Generative
RR +DNF [1]	Boolformula	No	DNF formula	No	No	Yes
IRM [2]	Relational	No	Co-Clustering	Unary	No	Yes
RL +Datalog [3]	Relational	Yes	Datalog	Yes	Yes	No
RL +FOL(T) [4]	Relational	Yes	None	Yes	Yes	No
RR +Datalog [5]	Relational	Yes	Datalog	Limited	Yes	Yes
RR +λ [6]	Functional	Yes+	λ calc.; single function	No	Limited	Yes
CL +EC [7]	Functional	Yes	Combinatory calc.	Yes	Primitive	Yes
IKM	Relational	Yes	Datalog	Yes	Yes	Yes

Table 4.1: A comparison of the features of various Bayes LOT models; see text for a detailed discussion. *Representation* is whether concepts are relations, functions, or boolean formulae (propositional). *Concept invention* describes whether the model allows for explicit representation of new concepts. In the case of relational LOTs, this refers to predicate invention. In case of functional LOTs, this refers to the definition of new functions. *Recursion* describes, roughly, whether a concept definition can refer to itself, either directly or via another concept. *Generative model* is whether the probabilistic model is generative, i.e., the model includes a description of a process by which samples from the distribution over LOT expressions can be generated.[1] Goodman et al. 2008a, [2] Kemp et al. 2006, [3] Kemp, Goodman, and Tenenbaum 2007; Katz et al. 2008, [4] Kemp, Goodman, and Tenenbaum 2008, [5] Ullman, Goodman, and Tenenbaum 2012, [6] Piantadosi, Tenenbaum, and Goodman 2012; Piantadosi, Tenenbaum, and Goodman 2010 [7] Dechter et al. 2013; Ellis, Dechter, and Tenenbaum 2015; Ellis et al. 2013.

4.2 Related Work

The Bayes LOT literature is relatively young and quite small. As such, there we have not yet developed a good way to classify Bayes LOT models or to delineate with precision the different dimensions along which they vary. It is not even clear where exactly the lines around this class of models should be drawn. Table 4.1 is our attempt to group and compare the models in the literature along important dimensions, but it should be clear that this is a provisional classification. The first three columns — Representation, First Order, and Syntactic restrictions — have to do with the characteristics of the language of thought formalism used in the model. The fourth and fifth columns — Predicate invention and Recursion — are capabilities of the model, i.e., the fifth column should be interpreted as answering the question, "Does this model allow a concept to be defined in terms of itself?" The last column — Generative — describes whether the probability distribution used in the model is specified using a generative process.

The Infinite Relational Model (Kemp et al. 2006) (IRM) is the least expressive LOT in the table. It is a co-clustering model, and, as such, there is no explicit language over which concepts are defined. Although it should perhaps not be considered an LOT model at all, it is motivated by the desire to model relational concept acquisition with an unbounded number of latent concepts; therefore, it shares important characteristics with several of the models in Table 4.1, including ours. The general idea behind the IRM is to describe a relation over entities as determined by a potentially unbounded number of latent clusterings of those entities. The IRM uses a non-parametric Bayesian prior over the clusterings of entities, so that a smaller number of clusters is preferable, but there is no *a priori* limit on the number of clusters in the model's posterior. The latent clusters inferred by the model can be seen as learned unary predicates; it is this aspect of the IRM that we borrow in the IKM. That said, the IRM is not a first-order formalism; it does not allow for quantification over variables. For example, statements of the form $\forall X, Y. a(X, Y) \Longleftrightarrow b(Y, X)$ cannot be expressed. Because of this, none of the case studies we look at in this work can be expressed by the IRM.

Several of the Bayes LOT models in 4.1 are based on the *Rational Rules* (RR) framework, which was introduced in Goodman et al. 2008a. Independently of the representation language itself, we take the RR framework to be the specification of a prior over LOT expressions by using a probabilistic context free grammar (PCFG), a probabilistic generative model of expressions in a language described by a context free grammar. Goodman et al. 2008a introduced a RR model in that paper, which we call RR+DNF, which specified a distribution over concepts as propositional Boolean formulae in disjunctive normal form. Note that the authors present it as a first-order language, by describing their representation language as a constrained fragment of FOL. Namely, the language consists of formulae of the form:

$$\ell(x) \Longleftrightarrow D(x), \tag{4.1}$$

where $D(x)$ is a DNF over the single variable x. In a related paper (Goodman et al. 2008b), they show how this can be extended to a single additional quantified variable:

$$\begin{aligned} \ell(x) &\Longleftrightarrow \exists y. D(x, y) && (4.2) \\ \text{or } &\Longleftrightarrow \forall y. D(x, y). && (4.3) \end{aligned}$$

While these latter forms may technically be first order formulae, the general approach is not developed beyond these highly restricted forms.

The other major influence of the RR+DNF model is the learning algorithm. Goodman et al. 2008a use a Metropolis-Hastings algorithm which is a biased random walk over the space of theories driven by what they term *subtree regeneration* proposals. Given the derivation tree of a given hypothesis expression, the Metropolis-Hastings proposal operator chooses a random node in the derivation tree and re-samples the subtree at that node from a PCFG prior. With minor variation, this algorithm –

Metropolis-Hastings with subtree regeneration proposals – is used for inference in the works that use the RR+λ and the RR+Datalog models, which we briefly describe next.

Piantadosi and colleagues applied the RR framework to study several specific domains of learning, and they did so by applying the framework to the space of functional programs. In order to provide a computational account of conceptual change, Piantadosi, Tenenbaum, and Goodman (2012) modeled how children acquire the "cardinality principle" – the knowledge that the successive numbers in the counting list correspond to successively increasing cardinalities. Their model used the RR framework over a grammar that generates functions in the lambda calculus; this allowed them to model how a child might learn a function from a set to a number. In separate work, Piantadosi, Tenenbaum, and Goodman 2010 studied models of how people might acquire concepts over sets of objects as functions over those sets. The Bayes LOT model they used for both these studies is essentially the same, and we call it RR+λ model. The use of the lambda calculus is very appealing, as it straightforwardly allows for higher-order functions, has a highly compositional nature, and — for these reasons – has long been seen as a promising representation of the semantics of natural language itself. We have used them ourselves in other work (see Dechter et al. 2013). Despite this, it is unclear how the lambda calculus can be used as a foundation of a general purpose knowledge representation system. Briefly, this is because we know of no workable proposals for how knowledge represented in the lambda calculus can support reasoning, other than just computing the functions themselves. That is, while both aforementioned papers show how can use the RR+λ model to learn functions from sets to the numbers of elements in those sets or to learn functions that classify sets of objects as within a concept or not, there is no concrete proposal about how a learning agent might use this knowledge in any way other than to compute these functions, because doing so would seem to involve practical reasoning with higher-order logics. That said, there have been proposals along these lines in the past, especially in the program languages literature, and research is ongoing (see, for example, Miller and Nadathur 2012).

In Table 4.1, models whose name includes RL are those in which the prior probability of an expression is a function of its "representation length," usually a count of the number symbols and variables in the expression. Essentially, a prior of this form is a log-linear model in which the features correspond to various counts of interest. Therefore, these are not generative models, in the sense that one cannot sample directly from these priors or explicitly compute the normalized probability of an expression under the model. One pattern to notice in Table 4.1 is that the RL type models allow for predicate invention, at least implicitly, and the RR type models do not, or do so in a limited way. The RL models "allow" for predicate invention because they do not seek to provide a generative story for how one might sample theories or programs that contain an unspecified or arbitrary number of invented predicates. However, a generative model is an essential tool for the computational cognitive scientists who wants to implement their model, for the generative process itself provides one with the means to sample from the prior (in the case of Monte Carlo style inference algorithms) or — as in the case of the algorithm presented in this chapter —

to use the prior distribution as a guide to searching the space of hypotheses. This is especially true in the context of discrete and high-dimensional hypothesis spaces, since inference techniques for log-linear models typically rely heavily on gradient information for optimization. One of the main contributions of the IKM is that it shows how to describe a generative Bayes LOT model that naturally accommodates predicate invention.

The two models in Table 4.1 that use Datalog — the representation formalism that the IKM uses — are the RL+Datalog and the RR+Datalog models. Of the two papers we have lumped together under the RL+Datalog heading – Kemp, Goodman, and Tenenbaum 2007 and Katz et al. 2008 – the IKM is more similar to the former. This is because the latter, along with the RR+Datalog model presented in Ullman, Goodman, and Tenenbaum 2012, makes an explicit distinction between "observable" predicates and "core" predicates, a distinction that we discuss in more detail in Section 4.5.

The RR+Datalog model is the most fully developed model in formalizing a generative model over Datalog knowledge bases and in offering a learning algorithm for that model. There are several concrete ways in which the IKM goes beyond this model. First, the RR+Datalog model allows for a fixed number of latent predicates whereas the IKM provides an unbounded number. Second, as with the other RR type models, the RR+Datalog model uses an Metropolis-Hastings plus subtree regeneration algorithm for inference. In contrast, the IKM is designed specifically to enable learning as abductive proof search; that is, we show that even though we employ an expressive first-order knowledge base formalism, we are not forced to use a generate-and-test search procedure for learning. Both in terms of numbers of clauses and numbers of concepts learned simultaneously, our work here significantly surpasses the previous work on the RR+Datalog model. Given that the hypothesis space is exponential in the number of clauses, pushing this dimension of difficulty is a fundamental challenge for these kinds of models. Finally, we generalize the application of Datalog knowledge base models and show that they apply not only to the learning of "laws" in intuitive theories, but also to some "procedural" kinds of knowledge. Namely, we show that the counting procedure that is modeled with the RR+λ model in Piantadosi, Tenenbaum, and Goodman 2012 can also be learned using the IKM. In doing so, we attempt to expand the domain of this kind of model's applicability.

We note, finally, that one Bayes LOT model has well surpassed any other in terms of the expressivity and generality of its language of thought. This is the one we call RL+FOL(T) and which is presented in Kemp, Goodman, and Tenenbaum 2008. The language of thought for this model is first order logic (FOL) with transitive closure (the T). While we postpone our discussion of the pros and cons of this choice for later (Section 4.5), we clarify the cell in the Table 4.1 which says that RL+FOL(T) admits recursion. First order logic, in the generality used in that paper, does not have an explicit notion of "definition." However, since that model allows arbitrary first-order formulae, one can simulate definitions using bi-implication (i.e. $\forall X, Y. C(x, y) \Longleftrightarrow (C(x-1, y) \vee \dots))$.

4.3 The Infinite Knowledge Base Model

In this section, we give a brief re-cap of the IKM and describe its application to problems of concept learning; for details, see Dechter and Tenenbaum in prep. The IKM is a generative model that, given a *knowledge base schema* Σ, specifies a joint distribution over knowledge bases K and the observed data sets $\mathcal{D}$ that are generated from them. An example knowledge base schema and a possible knowledge base generated from it is shown in Figure 4-1. A knowledge base, like the one on the right of the figure, is a collection of *clauses*, each of which is either a *rule* or a *fact*. A rule is an implication $H \leftarrow B$ where H (the *head*) is an atom (that is, a term consisting of a single predicate symbol applied to some arguments) and B (the *body*) is a list of atoms. A rule's presence means that if all the atoms in B are true, for some instantiation of its variables, then H is true for that same variable instantiation. A fact is simply a rule with an empty body, meaning that the atom is true unconditionally.

The knowledge base in Figure 4-1b shows some of the properties of this knowledge representation. It is *first-order*, which means that it allows for variables (written with uppercase letters) as arguments. Because a knowledge base semantically corresponds to a disjunction of its clauses and each clause is one possible reason to judge that the head of the clause is true, this formalism is *incremental*; any subset of a knowledge base's clauses is another well-formed knowledge base. Our formalism includes two kinds of symbols, *named* symbols and *unnamed* ones. The named symbols are those like MOTHER and PHILIP which refer to things in the world. The unnamed symbols correspond to predicates and objects that are invented by the learner and obtain their meanings merely by their usage within the knowledge base. We identify them by their arity (the number of arguments they take) and an integer index. In this paper, we maintain the convention that any symbol of the form $\ell_i^{(a)}$ is the ith unnamed symbol of arity a (note that when the arity can be inferred from the context we drop the superscript). Because of their usage as references to unobserved relations and objects, we refer to unnamed predicate symbols and unnamed constant symbols as *latent* predicates and constants, respectively. For example, in 4-1b, the first two clauses contain three latent predicates $\ell_1^{(1)}, \ell_2^{(1)}$, and $\ell_1^{(2)}$ which are meant to be interpreted as latent predicates that correspond to the concepts of FEMALE, MALE and FATHER, respectively.

Figure 4-1a shows an example knowledge base schema from which the knowledge base in Figure 4-1b could be generated. A knowledge base schema Σ consists of a collection of *metarules* (in the example r_1, r_2 and r_3) and, for each possible symbol arity, a potentially infinite set of symbols $\mathbf{S}$, where $S^{(a)}$ is the set of symbols, named and unnamed, of arity a. A metarule is a template for a clause in the knowledge base, where the existentially quantified variables are *metavariables*, and a clause is instantiated from a metarule by substituting symbols from $\mathbf{S}$ for its variables. For example, rule r_1 in 4-1b is generated by instantiating r_1 with the substitution $\{A \mapsto \text{MOTHER}, B \mapsto \ell_1^{(1)}, C \mapsto \ell^{(1)}\}$.

$$R = \left\{ \begin{array}{l} r_1 = \exists ABC.A(X,Y) \leftarrow B(X), C(X,Y) \\ r_2 = \exists ABC.A(X,Y) \leftarrow B(X), C(Y,X) \\ r_2 = \exists AX.A(X) \end{array} \right\},$$

and

$$\mathbf{S}^{(0)} = \{1, 2, 3, 4, \ell_1^{(0)}, \ell_2^{(0)}, \dots\}$$
$$\mathbf{S}^{(1)} = \{\ell_1^{(1)}, \ell_2^{(1)}, \dots\}$$
$$\mathbf{S}^{(2)} = \{mother, father, son, daughter, \ell_1^{(2)}, \ell_2^{(2)}, \dots\}.$$

(a) A knowledge base schema $\Sigma = (R, \mathbf{S})$.

$$\begin{array}{l} mother(X,Y) \leftarrow \ell_1(X), \ell_1(X,Y) \\ father(X,Y) \leftarrow \ell_2(X), \ell_1(X,Y) \\ daughter(X,Y) \leftarrow \ell_2(X), \ell_1(Y,X) \\ son(X,Y) \leftarrow \ell_2(X), \ell_1(Y,X) \\ \ell_1(philip).\ell_1(charles). \\ \ell_2(philip).\ell_2(charles). \\ \ell_1(philip, charles).\ell_1(philip, anne). \\ \ell_1(philip, charles).\ell_1(philip,). \end{array}$$

(b) A knowledge base which is an instantiation of Σ. Latent predicates $\ell_1^{(1}, \ell_2^{(1)}, \ell_1^{(2)}$ correspond to the concepts of *male*, *female*, and *parent* respectively.

Figure 4-1: An example knowledge base schema and a sample instantiation demonstrating the use of the latent predicates.

A prior distribution over knowledge bases We obtain a probability distribution over knowledge bases by specifying a stochastic generative process over *schema instantiations*, which are collections of metarules with their metavariables instantiated. This generative process is broadly defined by the following stochastic procedure:

1. sample the size of the knowledge base N, the number of its clauses, from a distribution over the natural numbers;

2. for each $i \in 1, \dots, N$, sample a metarule c_i from a distribution over the knowledge base schema's metarules;

3. to instantiate the metavariables in the resulting knowledge base, sample a sequence of symbols of the appropriate arities.

As is the case in the Rational Rules model, we want our prior over knowledge bases to have the property that we assign higher probability to knowledge bases with fewer clauses, and we want them to exhibit a "rich gets richer" dynamics known as *preferential attachment* over symbol and metarule choice. Therefore, we choose to sample the number of clauses from a geometric distribution, the clause metarules from a symmetric Dirichlet-Multinoulli (with $\alpha < 1$) and the symbols from a collection of Chinese Restaurant Processes (CRPs), one for each symbol arity. This Dirichlet-Multinoulli (Gelman et al. 2014) is a canonical example of a distribution that exhibits preferential attachment over a finite number of elements, and the CRP (Pitman 2006) is its generalization to an infinite number of elements.

A likelihood distribution over observed data In this work, we focus on learning from positive examples; we assume that the learner's data is set of true facts about the world. To keep the model as general as possible, we assume that the observed data is simply sampled uniformly without replacement from the set of true facts in a given context. So, for example, the model is learning about kinship relations, then the context might be all the people at a family gathering and the observed data is an arbitrary subset of all the true relations that hold among all the people at the gathering. The *data likelihood* – i.e. the probability of observing the data given some context and hypothetical knowledge base – is this inversely proportional to the number of facts that the knowledge base licenses. This is an implementation of the *size principle* (Tenenbaum 1999), the idea that a preference for restrictive hypotheses facilitates concept learning from positive examples.

4.3.1 Inference

In Bayes LOT models, the goal of the learner is to perform posterior inference over the model given the observed data. Performing inference in Bayes LOT models is exceptionally demanding. Where implementation of inference has been attempted, the approach has mainly been to use some form of stochastic search. However, because the hypothesis spaces involved are discrete and very high dimensional, and because expressions in LOT models are (like computer code) very fragile, the probability that a random perturbation to a hypothesis in a LOT moves to an improvement over the current one is vanishingly small.

There is also the psychological observation that human hypothesis generation does not seem to be a process of random exploration, that children and adults seems to have an ability to explore the space of possible explanations for their observations in a way that prunes the hypothesis space dramatically, and that accounting for this ability is key to understanding everyday conceptual discovery (see Schulz 2012 for some arguments along these lines).

Motivated by both the practical and theoretical limitations of learning algorithms based on stochastic search, we developed the HS-MAP-IKM algorithm (Dechter and Tenenbaum in prep.) – an algorithm for posterior inference in the IKM that searches over hypothetical knowledge bases in a targeted fashion. While the details of this algorithm are beyond the scope of this paper, we briefly describe the main ideas.

The algorithm performs *maximum a posteriori* inference: given a data set, it searches for the knowledge base that maximizes the joint probability of the data set and the knowledge base under the generative model described above. Using a recently developed AI learning framework called *metainterpretative learning* (MIL), the algorithm searches not over knowledge bases directly, but over possible explanations of the observed data, maintaining a collection of partial explanations of the data. Each partial explanation corresponds to a knowledge base, namely, that knowledge base containing all the rules used in the explanation. Moreover, for each partial explanation there is some set of things yet to be explained. The algorithm uses a *heuristic* function that provides an estimate of the cost (in terms of the negative log probability under the model) of the remaining explanation. The sum of this heuristic function plus the cost of the partial explanation so far is the signal that guides the algorithm through the hypothesis space, thus reducing the MAP inference problem to a problem in a *heuristic state-space search* (Edelkamp and Schrödl 2012; Pearl 1984).

4.4 Case studies

In this section, we examine our model's behavior in three domains of concept acquisition that have been studied in the literature on conceptual development and intuitive theories. We explore this behavior both as a function of the amount of data it receives and the time course of learning, and we discuss the parallels between its behavior and the empirical literature on concept acquisition.

General methods In each of these case studies, we we will describe a learning *scenario* which consists of the data observed by the learner and the learner's background knowledge. For each scenario, we run the HS-MAP-IKM algorithm with the model hyperparameters set to $\alpha = 1.0$, $\gamma = 0.8$, and $\omega = 5$. ω is a data multiplier, which means that the higher ω is, the more the model weights the data likelihood relative to the prior. The MAP inference algorithm is run in all cases until the target theory is found.

The figure accompanying each case study will consist of three parts parts. First, there is a plot showing the timecourse of the found theories. Second, we show the relative loglikelihood of the found theories as a function of the data multiplier ω. Third, we show the discovered theories themselves.

4.4.1 Magnetism

In this case study, we simulate a learner in a simplified magnetism domain, borrowed from Ullman, Goodman, and Tenenbaum 2012. In this learning scenario, there are twelve objects. Unbeknownst to the learner, four are magnets, four are magnetic (i.e. interact with magnets but not with each other) and four are non-magnetic. The learner witnesses the interactions diagrammed in Figure 4-2 in terms of the observed binary predicate INTERACTS. The learner is given no background knowledge about these objects.

Figure 4-3 shows our algorithm's learning behavior on this dataset. Over the course of the simulation, the learner discovers a sequence of five knowledge bases of increasing posterior probability. The first of these, found immediately, seems psychologically implausible: there are two distinct classes of objects ℓ_1 and ℓ_2, but they contain the exact same extension, namely, all the magnets and magnetic objects. The theory is quickly improved on by K_2 and K_3, which are variants of the idea that there are two distinct classes of objects that interact with each other and that this interaction is symmetric. K_4 is our intuitive theory about magnetic interaction, introducing a new rule. Taken together the three rules in K_4 say that there are magnets and magnetic objects, that those two classes interact, and that if something is a magnetic then it is also magnetic. Finally, K_5, which is slighly favored in negative log-probability over K_4, replaces the new rule in K_4 with the rule that magnets attract magnets. which simply states that any object interacts with any other object. Whether K_4 or K_5 is the more favored depends on the exact parameters of the prior distribution. On the one hand, K_4 is slightly more parsimonious in terms of symbols used. On the other hand, K_5 uses only three distinct types of *metarules* whereas K_4 uses four. Therefore, which gets favored depends on the strength of the "preferential attachment" characteristic of the prior distribution.

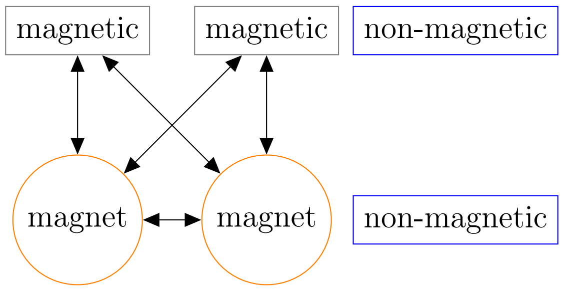


Figure 4-2: A schematic of the observed relations in the MAGNETISM experiment. In the actual experiment we use four objects of each type. The object labels are not known to the learner. According to this simplified theory of magnetic attraction, magnets interact with metals and other magnets (i.e. we do not take poles into account).

The magnetism case-study is a simple example of how the IKM is able to perform theory-learning as a kind of dimensionality reduction, using predicate invention to discover categories of objects that are not in the data to begin with, but, when introduced, provide a more parsimonious theory.

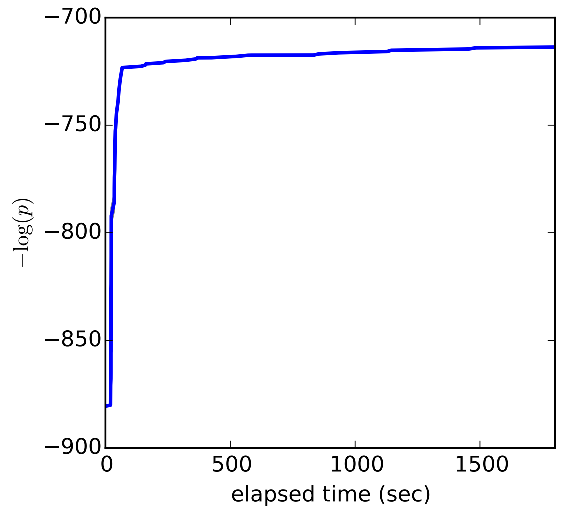
4.4.2 Kinship

Human kinship relations are a paradigmatic case of a relational system of concepts. The kinship domain features several important properties of systems of conceptual knowledge. Lexical kinship concepts are referred to by common words (like *mother*) are defined in terms of *core* concepts whose associated words are much less common in everyday speech (like *female* and *parent*).

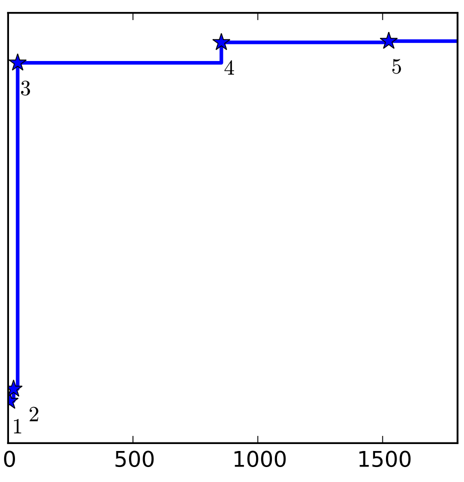
We consider the four binary kinship relations MOTHER, FATHER, SON, and DAUGHTER. Ideally, we might want to show that our model can learn the definitions of these four relations in terms of the core relations of MALE, FEMALE and PARENT. To keep the problem computationally manageable, however, we provide the MALE and FEMALE predicates as background knowledge and examine the extent to which the IKM can recover the definitions of the kinship relations as well as discover the PARENT/2 relation as a latent predicate.

Methods The graphs shown in Figure 4-4 show a fragment of the family tree of the British royal family. Figure 4-4a shows the core unary predicates *male* and *female* across two generations. These are provided as background knowledge. Figure 4-4b shows the true observed relations between these individuals, i.e. the data given to the learner.

Results Figure 4-5 shows the timecourse of the learning algorithm, with corresponding found knowledge bases shown in Figure 4-6. K_1 is all data memorized in the knowledge base. Knowledge bases K_2 through K_5 increasingly substitute rules for some of these memorized facts, though none use any latent predicates for this. Finally, K_6 makes a leap to using a latent binary predicate, whose extension which is equivalent to $child/2$.



a) The average negative log-probability of discovered theories as a function of learning time. Error-band is the standard deviation.



b) A single representative run. Stars correspond to discovered theories. The numbered knowledge bases correspond to these stars.

K_1

$attracts(A, B)$
$\leftarrow \ell_2(A), \ell_3(B)$

$\ell_2(f_1)$ $\ell_3(m_2)$
$\ell_3(f_4)$ $\ell_2(f_3)$
$\ell_2(f_4)$ $\ell_2(m_2)$
$\ell_3(m_1)$ $\ell_3(m_3)$
$\ell_3(m_4)$ $\ell_2(m_4)$
$\ell_3(f_3)$ $\ell_2(f_2)$
$\ell_3(f_1)$ $\ell_3(f_2)$
$\ell_2(m_1)$ $\ell_2(m_3)$

K_2

$attracts(A, B)$
$\leftarrow \ell_2(A), \ell_3(B)$
$attracts(B, A)$
$\leftarrow attracts(A, B)$

$\ell_3(f_3)$ $\ell_3(f_4)$
$\ell_3(f_1)$ $\ell_2(f_2)$
$\ell_2(m_1)$ $\ell_3(f_2)$
$\ell_3(m_3)$ $\ell_2(m_2)$
$\ell_3(m_2)$ $\ell_2(m_4)$
$\ell_2(f_4)$ $\ell_3(m_1)$
$\ell_2(f_3)$ $\ell_2(f_1)$

K_3

$attracts(A, B)$
$\leftarrow \ell_2(A), \ell_3(B)$
$attracts(B, A)$
$\leftarrow attracts(A, B)$

$\ell_3(m_1)$ $\ell_3(m_4)$
$\ell_2(f_1)$ $\ell_3(m_3)$
$\ell_2(m_4)$ $\ell_2(f_2)$
$\ell_3(m_2)$ $\ell_2(f_4)$
$\ell_2(f_3)$ $\ell_2(m_3)$
$\ell_2(m_2)$

K_4

$attracts(A, B)$
$\leftarrow \ell_2(A), \ell_3(B)$
$attracts(B, A)$
$\leftarrow attracts(A, B)$
$\ell_3(A)$
$\leftarrow \ell_2(A)$

$\ell_2(m_1)$ $\ell_3(f_1)$
$\ell_2(m_2)$ $\ell_3(f_2)$
$\ell_2(m_3)$ $\ell_3(f_3)$
$\ell_2(m_4)$ $\ell_3(f_4)$

K_5

$attracts(A, B)$
$\leftarrow \ell_3(A), \ell_2(B)$
$attracts(A, B)$
$\leftarrow \ell_2(A), \ell_2(B)$
$attracts(B, A)$
$\leftarrow attracts(A, B)$

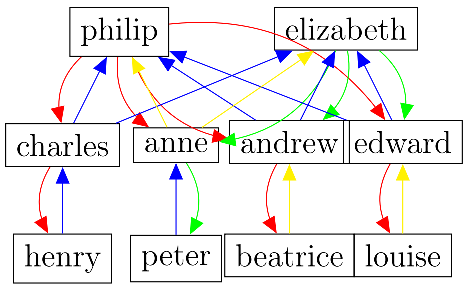
$\ell_2(m_3)$ $\ell_3(f_1)$
$\ell_3(f_2)$ $\ell_2(m_1)$
$\ell_2(m_2)$ $\ell_3(f_4)$
$\ell_2(m_4)$ $\ell_3(f_3)$

c) Knowledge bases for discovered theories in a representative run of the magnetism experiment.

Figure 4-3: Results of the MAGNETISM experiment. In this learning task, there are four magnets ($m_1, \ldots, m_4$), four magentic objects $f_1, \ldots, f_4$), and four non-magnetic objects ($n_1, \ldots, n_4$). Note, however, that the naming of these symbols is just for the

Male	Female
philip	elizabeth
charles	anne
henry	beatrice
andrew	louise
edward	
peter	

(a) Background knowledge: unary predicates male/1 and female/1.



(b) Observable binary relations: FATHER (red), MOTHER (green), SON (blue), DAUGHTER (yellow).

Figure 4-4: Background and observed relations for KINSHIP case study. Whereas in the MAGNETISM case-study, the learner invents unary predicates to explain the observed relation, in the KINSHIP case-study, the learner invents a binary-predicate (PARENT/2) to explain the relation between observed kinship terms and gender.

4.4.3 Counting

In our final case study, we model the acquisition of the knowledge about the relationships between number words and cardinalities. How children come to understand the meaning of numbers and their relationship to cardinalities is a problem with a long history and continuing debate in the literature on conceptual development (see Wynn 1992; Sarnecka and Carey 2008; Leslie, Gelman, and Gallistel 2008). As discussed earlier, Piantadosi, Tenenbaum, and Goodman 2012 used a Bayes LOT model as a model of the the acquisition of the CP principle. In that work, the LOT is the lambda calculus and the goal is to learn a function from sets of objects to number words.

The Piantadosi et almodel explains children's progression through the N-knower stages entirely at the computational level. Under the model's prior, the prior probability decreases as the learner transitions from 0-knower to 1-knower, 1-knower to 2-knower, and so on. On the other hand, the data likelihood increases along these same transitions. The balance between these two factors determines the optimal theory. As the amount of data increases, the influence of the data likelihood increases, and so the optimal theory moves further towards the CP-knower. Therefore, the learning trajectory is determined by the amount of the data the learner observes.

There are two unsatisfying aspects to the Piantadosi model:

1. in order to match the empirically observed learning trajectory, the model needs to impose a large log-probability penalty on the use of recursion. This is because the CP-knower theory can be expressed relatively concisely when recursion is employed. Figure 4-7 shows the 3-knower and CP-knower theories as expressed in the Piantadosi model; the size of the 3-knower theory is on-par if not somewhat greater than that of the CP-knower theory. In order to get the learner to progress through a 3-knower stage, then, it is necessary that the recursive call (the L symbol in the figure) be less likely under the prior than other symbols and productions in the LOT grammar.

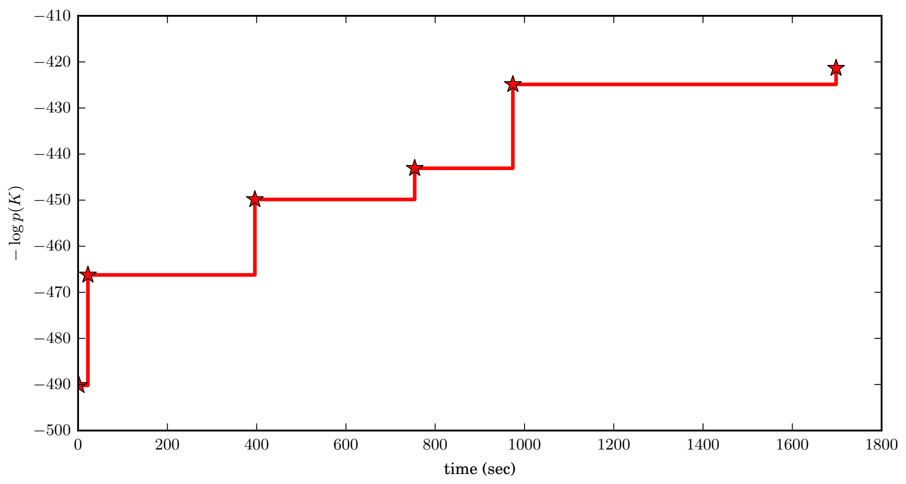


Figure 4-5: Timecourse of log probability of best kinship theory found as a function time. The knowledge bases corresponding to these discovered theories are shown in Figure 4-6.

2. More importantly, the Piantadosi model's reliance on computational level considerations to explain the learning trajectory does not accord with our intuition that the learning trajectory should be strongly influenced by the process which implements that learning. On the basis of this model, one might expect a child who is quickly exposed to a large amount of data about sets and their cardinalities to skip the N-knower stages altogether or to progress through them with arbitrary speed. An algorithmic level model does not have this same problem, because it credits the learning trajectory to the algorithmic process by which learning is accomplished. Doing so has the additional benefit of solving the previous problem: if the model's explanatory power does not solely depend on the relative probabilities of the various theories, it becomes more robust to the exact details of the prior distribution over expressions.

Our model addresses these aspects by attempting to account for children's observed learning trajectories through the search process the child goes through in attempting to find theories that best explain their observations.

Learning scenario The simulated learning scenario for this case study is similar to that in Piantadosi, Tenenbaum, and Goodman 2012. The learner is assumed to observe sets and hear number words corresponding to their cardinalities. Following Piantadosi et al., who examined number word frequencies in the CHILDES corpus, we use a distribution over set sizes that is strongly biased towards "one," but that has a heavy tail up to "five." This distribution is shown in Figure 4-9.

The learning scenario for this case study is shown in Table 4-10. The data set consists of two "episodes" in which the learner sees four sets paired with the corresponding number word; the constants $s_{i,j}$ correspond to the set in episode i with j

$daughter(anne, elizabeth)$
$son(edward, elizabeth)$
$father(edward, louise)$
$father(charles, william)$
$daughter(louise, edward)$
$father(philip, edward)$
$mother(elizabeth, edward)$
$mother(elizabeth, charles)$
$son(charles, elizabeth)$
$son(peter, anne)$
$daughter(beatrice, andrew)$
$son(andrew, elizabeth)$
$father(philip, anne)$
$father(philip, andrew)$
$mother(elizabeth, andrew)$
$mother(anne, peter)$
$father(andrew, beatrice)$
$mother(elizabeth, anne)$
$son(andrew, philip)$
$son(charles, philip)$
$son(william, charles)$
$father(philip, charles)$
$daughter(anne, philip)$
$son(edward, philip)$

(a) K_1

$mother(D, E) \leftarrow female(D), son(E, D)$
$son(peter, anne)$
$daughter(louise, edward)$
$son(charles, elizabeth)$
$father(charles, william)$
$father(philip, edward)$
$father(philip, charles)$
$son(edward, philip)$
$daughter(anne, philip)$
$mother(elizabeth, anne)$
$son(edward, elizabeth)$
$son(andrew, philip)$
$daughter(anne, elizabeth)$
$son(charles, philip)$
$daughter(beatrice, andrew)$
$father(philip, anne)$
$father(edward, louise)$
$son(andrew, elizabeth)$
$father(philip, andrew)$
$father(andrew, beatrice)$
$son(william, charles)g$

(b) K_2

$daughter(D, E) \leftarrow female(D), father(E, D)$
$father(I, J) \leftarrow male(I), son(J, I)$
$mother(elizabeth, edward)$
$son(andrew, elizabeth)$
$son(william, charles)$
$son(charles, elizabeth)$
$father(edward, louise)$
$son(edward, philip)$
$son(andrew, philip)$
$daughter(anne, elizabeth)$
$son(edward, elizabeth)$
$mother(anne, peter)$
$father(philip, anne)$
$mother(elizabeth, anne)$
$mother(elizabeth, andrew)$
$father(andrew, beatrice)$
$son(charles, philip)$
$son(peter, anne)$
$mother(elizabeth, charles)$

(c) K_3

$mother(D, E) \leftarrow female(D), son(E, D)$
$father(I, J) \leftarrow male(I), son(J, I)$
$son(andrew, elizabeth)$
$mother(elizabeth, anne)$
$father(philip, anne)$
$son(edward, elizabeth)$
$son(charles, elizabeth)$
$son(charles, philip)$
$son(peter, anne)$
$son(edward, philip)$
$father(andrew, beatrice)$
$daughter(anne, elizabeth)$
$daughter(anne, philip)$
$daughter(beatrice, andrew)$
$son(andrew, philip)$
$son(william, charles)$
$daughter(louise, edward)$
$father(edward, louise)$

(d) K_4

$mother(D, E) \leftarrow female(D), son(E, D)$
$daughter(I, J) \leftarrow female(I), father(J, I)$
$father(K, L) \leftarrow male(K), son(L, K)$
$son(andrew, philip)$
$son(edward, philip)$
$mother(elizabeth, anne)$
$son(peter, anne)$
$son(andrew, elizabeth)$
$son(edward, elizabeth)$
$son(charles, philip)$
$daughter(anne, elizabeth)$
$son(charles, elizabeth)$
$son(william, charles)$
$father(edward, louise)$
$father(andrew, beatrice)$
$father(philip, anne)$

(e) K_5

$son(A, B) \leftarrow male(A), \ell_1(A, B)$
$father(I, J) \leftarrow male(I), \ell_1(J, I)$
$mother(N, O) \leftarrow female(N), \ell_1(O, N)$
$daughter(P, Q) \leftarrow female(P), \ell_1(P, Q)$
$\ell_1(charles, philip)$
$\ell_1(charles, elizabeth)$
$\ell_1(anne, philip)$
$\ell_1(peter, anne)$
$\ell_1(edward, philip)$
$\ell_1(william, charles)$
$\ell_1(beatrice, andrew)$
$\ell_1(andrew, elizabeth)$
$\ell_1(louise, edward)$
$\ell_1(andrew, philip)$
$\ell_1(anne, elizabeth)$
$\ell_1(edward, elizabeth)$

(f) K_6

Figure 4-6: Learned theories in the KINSHIP case-study, in order of increasing posterior log probability. Figure 4-5 shows the timecourse of search. The learner is able to directly observe predicates MALE/1, FEMALE/1, MOTHER/2, FATHER/2, SON/2, and DAUGHTER/2. Theoresi K_1 through K_5 make due with these predicates, refining the relations between them in order to create more parsimonious theories. But the optimal theory K_6 is created by inventing a new binary predicate $\ell_1/2$ which corresponds to the PARENT/2 relation.

THREE-KNOWER

```
(lambda S (if (singleton? S)
              "one"
           (if (doubleton? S)
               "two"
             (if (tripleton? S)
                 "three"
               undef))))
```

(a)

CP-KNOWER

```
(lambda S (if (singleton? S)
              "one"
           (next (L (set-difference S
                                       (select S))))))
```

(b)

Figure 4-7: The THREE-KNOWER and CP-KNOWER expressions in the Piantadosi et al. model.

elements. For example, the observed fact CARD($s_{2,4}$,“four") corresponds to the observation that the cardinality of the set with four elements in the second episode is number word “four.”

The learner is also given a collection of background knowledge predicates. NUM/1 and SET/1 are predicates that hold of any number word and set, respectively. NEXT/2 corresponds to knowledge of successive element in the number word count list (“one”, “two”, . . .). ADD_ONE/2 relates a set to the result of adding an element to that set, e.g., ADD_ONE($s_{1,2}, s_{1,3}$) holds because the learner observes that $s_{1,3}$ resulted when a single element was added to $s_{1,2}$. REMOVE_ONE/2 is the inverse of ADD_ONE/2. The predicates SINGLETON/2, PAIR/2, TRIPLE/2, and MANY/2 are predicates that correspond to the empirical observations that children have the ability to distinguish sets of one, two, and three elements from one another and larger sets, even before they learn how to count.

Our knowledge base formalism is quite different from that used in Piantadosi, Tenenbaum, and Goodman 2012; the IKM learns first-order knowledge bases. While not every function expressible in the lambda calculus can be represented in our first-order knowledge bases, many, including the one learned Piantadosi, Tenenbaum, and Goodman 2012, can be, and in a very similar way.

Piantadosi, Tenenbaum, and Goodman 2012 modeled a child learning a function from sets to number words. In their formalism the CP-KNOWER’s knowledge is expressed by the function in Figure 4-7b. To write this in our knowledge base representation we define instead binary predicate CARD which is true whenever its first argument is a set and its second argument is the number word corresponding to the set. Since the functional version has a single conditional, the relation version is a disjunction of two clauses:

$$\begin{aligned}&\textsc{card}(S, N) \leftarrow \textsc{card}(S', N'), add\_one(S', S), next(N', N).\\&\textsc{card}(S, \text{“one”}) \leftarrow singleton(S).\end{aligned} \tag{4.4}$$

Note that while the functional and relation versions are equivalent when interpreted as functions from set to numbers, the relational version can be used to answer queries in the opposite direction as well, e.g., we can query the knowledge base for those sets S such that $\textsc{card}(S, \text{“two”})$. Thus in the relational formalism we do not need separate knowledge for counting the number of elements in a set and providing a set with a specified number of elements.

We ran multiple runs ($N = 20$) of simulated learning of the IKM in this learning scenario. Stochasticity across runs is due to a stochastic goal selection function, which can be interpreted as a kind of stochastic “attention” or randomness in the order in which data are seen. Each of the runs had a time limit of one hour. We chose to use a data set of size 200 because, as is explained in the next section, Figure 4-8a this seems to be the smallest data-set size that makes N-knower progression possible. (This is because our learning algorithm, begin a MAP learning algorithm, never moves from a one theory to a lower probability theory, and so we needed to choose a data set size at which the probability along the N-knower progression is strictly increasing).

Results We first confirm that under our model and the given learning scenario, a computational level analysis alone does not account of for the progression of N-knower stages. To do this we manually constructed five knowledge bases corresponding to these stages (Figure 4-8b) and examined their probabilities under the model as a function of the amount of data observed. Figure 4-8a shows the negative log probability of these five knowledge bases as the amount of data increases. As the amount of data increases, the maximum amount the first four knowledge bases progressively moves to the right, but, regardless of the amount of data, the overall best knowledge base is the CP-KNOWER one.

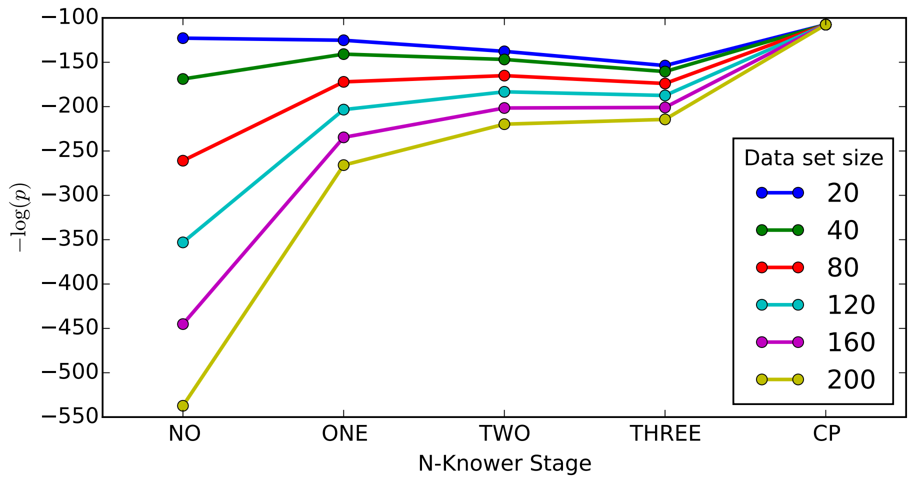
Note, also, that in our formalism, it is less clear how one would apply a "recursion penalty" of the form used in the Piantadosi model, because the the presence of the recursion is not a local property of a single clause, but is a property of the knowledge base as a whole. In fact, the our hand-written knowledge base for the CP-KNOWER does not involve any directly recursive clauses (clauses whose head predicate appears in the body).

Figure 4-11 shows the results from the multiple simulated learning runs in the counting scenario. Figure 4-11(a) shows the mean loglikelihood of the current best knowledge base as a function of time. We show only a portion of the hour long run time because the mean function quickly plateaus. This data is broken down by N-knower stage in Figure 4-11(c), in which we test each learner's theory at each point in time to see if it corresponds to one the N-knower stages. The curves in the figure correspond to the percentage of learners in a given stage at a given point in time.

Note that even though nearly 90% of the learners are CP-KNOWERS by the end of the simulation, the negative log probability of their knowledge bases is not as high as that shown for the CP-KNOWER in Figure 4-8a. This is because there are many knowledge bases that are extensionally equivalent to our manually constructed CP-KNOWER knowledge base, but are not as concise.

The inference algorithm discovers three knowledge bases K_1, K_2 and K_3. K_1 is found immediately and is the theory that any number word can be paired with any set. K_2 is an intermediate theory that sets with one and two elements have cardinality "one" and "two," respectively, but that beyond two elements, any number word can be used. This corresponds to the 2-knower stage in the conceptual development of counting concepts. The last, K_3, is the full CP-knower theory, equivalent to that shown in Equation 4.4, but using a binary latent predicate to fit a clause with three atoms in its body into a knowledge base schema confined to clauses with bodies of at most two atoms. This is an example of how latent predicates can be used both to express latent core concepts and as intermediate concepts that facilitate fitting complex expressions into a simple formalism.

In Piantadosi, Tenenbaum, and Goodman 2012, the authors argue that children's progression through the N-knower stages can be accounted for purely by considerations at the "computational" level of the model specification. The distribution of the number words that children hear is strongly biased towards "one." The authors argue that this, combined with a hypothetical probability penalty for recursion, might drive the step-wise progression that children go through. While we do not have evidence against this view, our results show a similar progression through intermediate theories of cardinality as a byproduct of the search dynamics, suggesting perhaps that stages of development might be driven by considerations at the "algorithmic" level of analysis. That is, the progression between N-knower stages might be due to the structure of the search space itself. On this view, it is not that 2-knowers, say, have in some sense considered the CP-knower hypothesis and rejected it; it is that they simply have not found it yet.



(a) Negative log probability of N-knower KBs as a function of the amount of observed data. The KBs corresponding to the x-axis labels are in Figure 4-8b.

NO-KNOWER

$\mathrm{card}(A, B) \leftarrow set(A), num(B)$

ONE-KNOWER

$\mathrm{card}(A, B) \leftarrow single(A), \ell_1(B)$

$\ell_1(\mathrm{one})$

$\mathrm{card}(A, B) \leftarrow \mathrm{more\_than\_single}(A),$
$\quad l_1(A, B)$

$\ell_1(A, B) \leftarrow \mathrm{set}(X), \mathrm{num}(Y)$

TWO-KNOWER

$\mathrm{card}(A, B) \leftarrow single(A), \ell_1(B)$

$\ell_1(\mathrm{one})$

$\mathrm{card}(A, B) \leftarrow pair(A), \ell_2(B)$

$\ell_2(\mathrm{two})$

$\mathrm{card}(A, B) \leftarrow \mathrm{more\_than\_pair}(A),$
$\quad l_1(A, B)$

$\ell_1(A, B) \leftarrow \mathrm{set}(X), \mathrm{num}(Y)$

THREE-KNOWER

$\mathrm{card}(A, B) \leftarrow single(A), \ell_1(B)$

$\ell_1(\mathrm{one})$

$\mathrm{card}(A, B) \leftarrow pair(A), \ell_2(B)$

$\ell_2(\mathrm{two})$

$\mathrm{card}(A, B) \leftarrow triple(A), \ell_3(B)$

$\ell_3(\mathrm{three})$

$\mathrm{card}(A, B) \leftarrow \mathrm{more\_than\_triple}(A),$
$\quad l_1(A, B)$

$\ell_1(A, B) \leftarrow \mathrm{set}(X), \mathrm{num}(Y)$

CP-KNOWER

$\mathrm{card}(A, C) \leftarrow \ell_1(A, B), \mathrm{next}(B, C)$

$\ell_1(A, C) \leftarrow \mathrm{remove\_one}(A, B), \mathrm{card}(B, C)$

$\mathrm{card}(A, B) \leftarrow \mathrm{single}(A), \ell_1(Y)$

$\ell_1(\mathrm{one})$

(b) Manually constructed KBs for the N-knower stages.

Figure 4-8: As is the case for the Piantadosi model of the the N-knower progression, our a straightforward implementation of the N-knower KBs in our model does not explain the progression at a computational level.

.

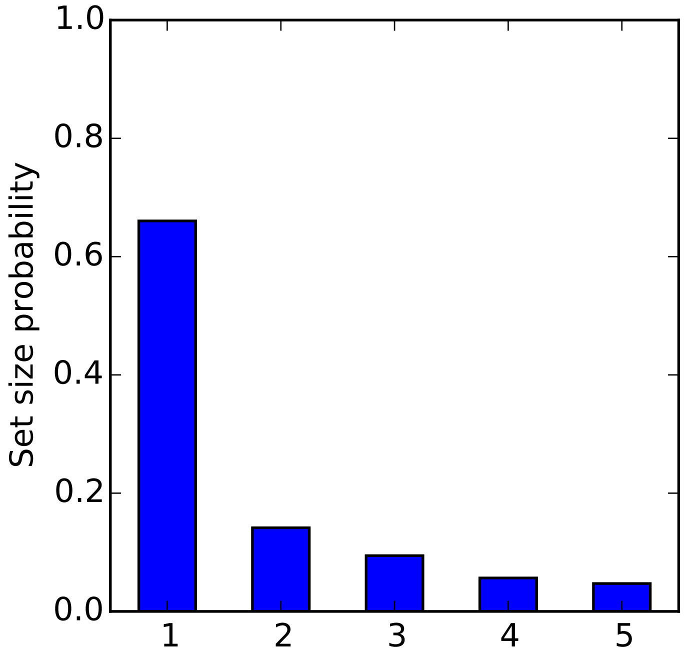


Figure 4-9: Set size frequencies used to simulate data for counting experiment.

Background Knowledge	
NUM/1	NUM(N) iff N is a number word
SET/1	SET(S) iff S is a set
SINGLETON/1	SINGLETON(S) iff S is a singleton set
PAIR/1	PAIR(S) iff S is a set of size two
TRIPLE/1	TRIPLE(S) iff S is a set of size three
MANY/1	MANY(S) iff S is a set of size greater than three
SUCC/2	SUCC(one, two), SUCC(two, three), ...
ADD_ONE/2	ADD_ONE(S_1, S_2) iff S_1 is the set obtained by adding a single elements to S_2.
REMOVE_ONE/2	REMOVE_ONE(S_1, S_2) iff S_1 is the set obtained by removing a single elements to S_2.
Observed Predicates	
CARD/2	CARD($Set, Number$) where $Number$ is the cardinality of set Set.

Observed data
CARD($s_{1,1}, one$).CARD($s_{1,2}, two$).
CARD($s_{1,2}, three$).CARD($s_{1,4}, four$).
CARD($s_{2,1}, one$).CARD($s_{2,2}, two$).
CARD($s_{2,3}, three$).CARD($s_{2,4}, four$).

Figure 4-10: Learning scenario for counting case study.

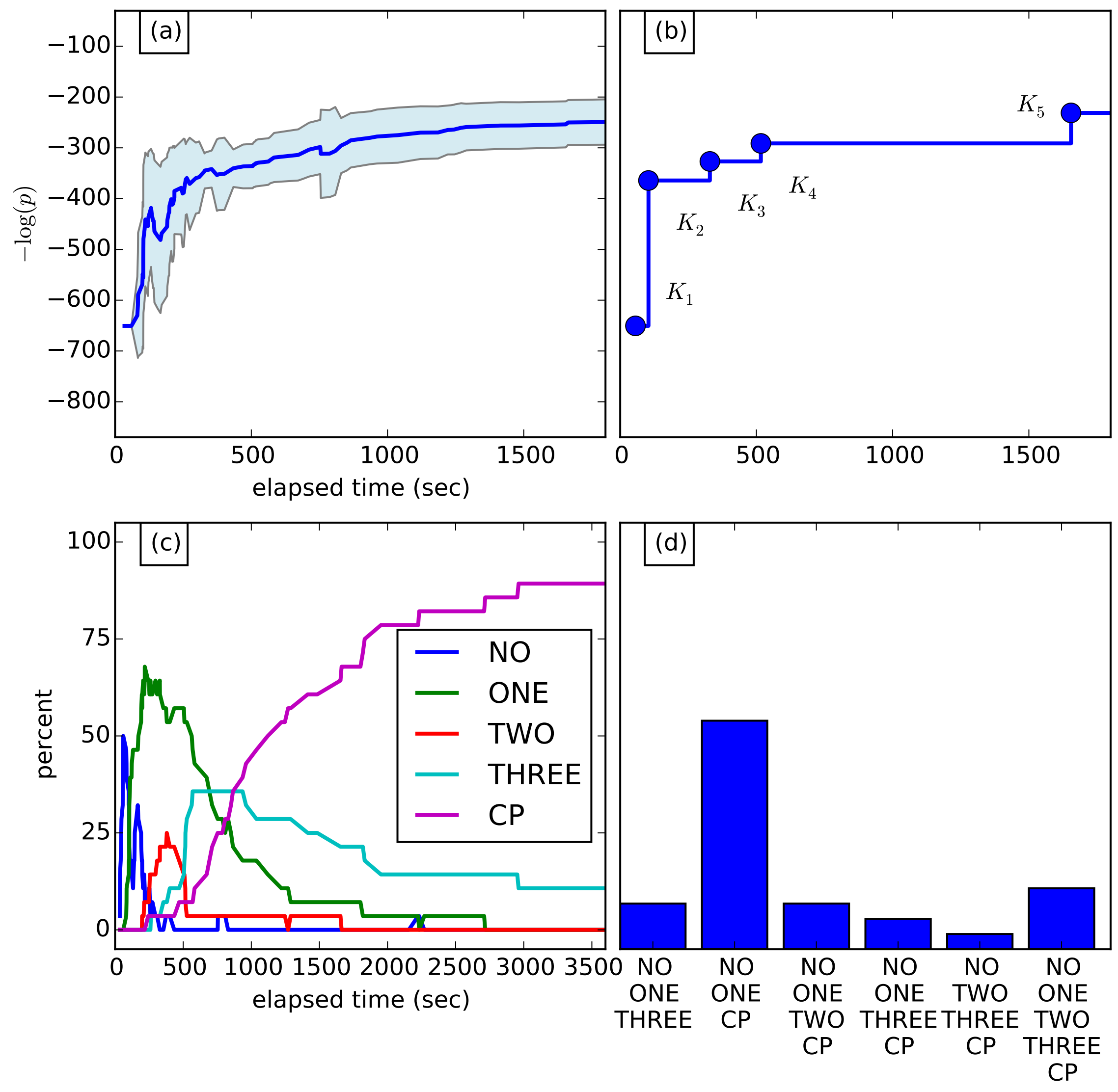


Figure 4-11: Results from the COUNTING case study. (a) Average negative loglikelihood of the learned theory as a function of time. The ribbon shows one standard deviation. (b) An example of a learner who goes through all the N-knower stages. The plot shows the negative loglikelihood of the theories $K_1, \ldots, K_5$ which correspond to learning stages NO-KNOWER, . . . , CP-KNOWER. The knowledge bases are shown in Figure 4-12. (c) The percentage of learners at N-knower stage at a given point in time. Note that sum of curves need not sum to 100 since early on a learner may not have discovered any theory at all. (d) The percentage of learners in each of the learning trajectories occurring in the data.

K_1

$card(A, B)$
$\leftarrow set(A), num(B)$

K_2

$card(A, B)$
$\leftarrow single(A), \ell\_9(B)$
$card(A, B)$
$\leftarrow more\_than\_single(A), num(B)$
$\ell\_9(one)$

K_3

$\ell\_5(A, B)$
$\leftarrow more\_than\_pair(A), \ell\_10(B)$
$card(A, B)$
$\leftarrow single(A), \ell\_9(B)$
$card(A, C)$
$\leftarrow add\_one(A, B), \ell\_5(B, C)$
$card(A, B)$
$\leftarrow more\_than\_pair(A), num(B)$
$\ell\_10(two)$
$\ell\_9(one)$

K_4

$card(A, B)$
$\leftarrow triple(A), \ell\_11(B)$
$card(A, B)$
$\leftarrow more\_than\_triple(A), num(B)$
$card(A, B)$
$\leftarrow single(A), \ell\_10(B)$
$card(A, B)$
$\leftarrow pair(A), \ell\_9(B)$
$\ell\_11(three)$
$\ell\_9(two)$
$\ell\_10(one)$

K_5

$\ell\_5(A, B)$
$\leftarrow \ell\_9(A), single(B)$
$card(A, C)$
$\leftarrow \ell\_5(A, B), add\_one(B, C)$
$\ell\_5(B, A)$
$\leftarrow card(A, B)$
$card(A, C)$
$\leftarrow add\_one(A, B), \ell\_5(B, C)$
$card(A, C)$
$\leftarrow \ell\_5(A, B), next(B, C)$
$\ell\_9(one)$

Figure 4-12: KBs learned in the run of COUNTING case study shown in Figure 4-11(b). This learner progresses in sequences through the N-knower stages: K_1 corresponds to NO-KNOWER, K_2 to ONE-KNOWER, K_3 to TWO-KNOWER, K_4 to THREE-KNOWER,K_5 to CP-KNOWER.

4.5 Discussion

In this paper we have shown the IKM can be used to model several of the kinds of systems of concepts that children learn over the first few years of life. As a whole, the IKM unifies and generalizes many of the previous approaches to Bayes LOT models, but there are also several ways in which it contrasts with them.

Uniform treatment of rules and "core" predicates Ullman, Goodman, and Tenenbaum 2012 and Katz et al. 2008 make a formal distinction between "core" predicates and "observable" predicates. Core predicates cannot appear as the consequence of a rule and are represented as completely specified relations, e.g., the PARENT/2 core predicate is a boolean table over all pairs of entities in the domain. Given that "unexplained" knowledge can simply be included as facts in the knowledge base, it is not clear what this distinction buys representationally. Moreover, it seems likely that core predicates are only provisionally so, until enough data has been observed and enough search has been done to discover predicates that are more "core" than the ones discovered so far. However, there may be algorithmic benefits to treating the search over facts differently from the search over rules, especially in the regime of larger amounts of data.

Functional vs relational representations We have already discussed how the specific concepts studied in Piantadosi, Tenenbaum, and Goodman 2012 using a lambda calculus representation can be equivalently represented in our knowledge base formalism. There are, however, several advantages to using a representation based on a functional language. First, in the case where the target relation is a function, using a functional language reduces the hypothesis space to only those that satisfy this property. Second, a functional language naturally allows for higher-order functions (e.g. a function that takes another function as an argument and applies it to every element in a set). While both of these properties could potentially be simulated in a logic programming language, it is not clear whether the inference algorithms we have developed will be useful in these cases.

Relatedly, as we pointed out in our discussion of K_3 in the magnetism case study, there are often higher-order properties of relations that one would either want to learn or take as an assumption. In that example, the higher-order property was that any definition of the predicate INTERACTS/2 should be symmetric in its arguments. That a predicate should be functional in one of its arguments (as in the example CARD/2, in which the number word is a function of the set) is another such example. Type level information (for example, that a given argument of a predicate must be a word and not a physical object) is also a kind of higher-order specification. It is not clear exactly where such higher-order properties should be learned or specified in a concept learning model. Some of these properties, like symmetry, we can simulate in our knowledge base formalism: e.g., if we want to specify that INTERACTS/2 is symmetric we can just posit that INTERACTS$(X, Y) \leftarrow$ INTERACTS(Y, X) is the in the knowledge base to begin with. But for others, like the functionality constraint, cannot be expressed in this way.

Negation The most glaring limitation of the model proposed here is that in its current form it does not implement any form of explicit negation. The semantics of the knowledge base is based on the *closed world assumption*, the assumption that any fact that cannot be proven using the knowledge base is false. In logic programming and deductive databases, a form of explicit negation called *negation as failure* is often used to allow for clauses that include negated atoms in their bodies. Under negation as failure, $\neg a(x)$ is true when $a(x)$ is not in the extension of the predicate $a/1$. This allows one to write rules of the form

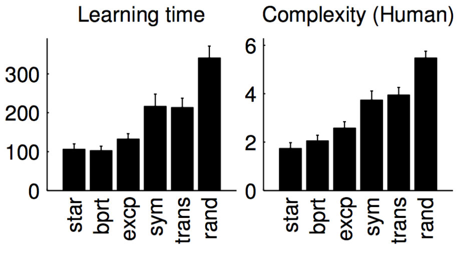
$$\textsc{orphan}(\mathrm{X}) \leftarrow \neg\textsc{parent}(\mathrm{Y},\ \mathrm{X}), \tag{4.5}$$

which reads that someone is an orphan if there is no person who is their parent. These are concepts that people can clearly express, and so any serious attempt to model people's conceptual learning must be able to account for it.

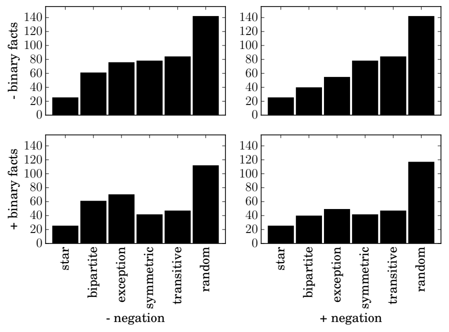
In the IKM, such negation-as-failure within each hypothetical knowledge would take, from the perspective of the learner, the form of what we call *negation-as-unabduced*; by this we mean that some atom is considered false, if the learner is unwilling to abduce the clause or clauses that would make it true. To take the example in clause 4.5 above, suppose the learner is attempting to learn a knowledge base that entails observed data of $\textsc{orphan}/1$ given some potentially incomplete set of $\textsc{parent}/2$ facts, and suppose one of the observations is $\textsc{orphan}(john)$. Even if there is no parent of John mentioned in the knowledge base, the learner can always use an unnamed symbol – say, ℓ_1 – to stand in for John's parent. Therefore, being able to prove that John is an orphan using clause 4.5 depends on the learner's unwillingness to abduce the fact $\textsc{parent}(\ell_1, john)$.

There are a couple different reasons we have not included negation-as-failure in our implemented model. The first is that the technology that we have used to implement the generated knowledge bases does not yet handle negation. (As of version 7.3.34, the tabling extension in SWI-Prolog has not implemented negation.) The second reason is that handling negation-as-failure in our inference algorithm would require a significant amount of complication and modification, and would likely require substantial changes to the way in which we compute our heuristic function. Given the computational challenges of exploring the space of knowledge bases, we decided to tackle the case in which negation is not used before addressing the more general case.

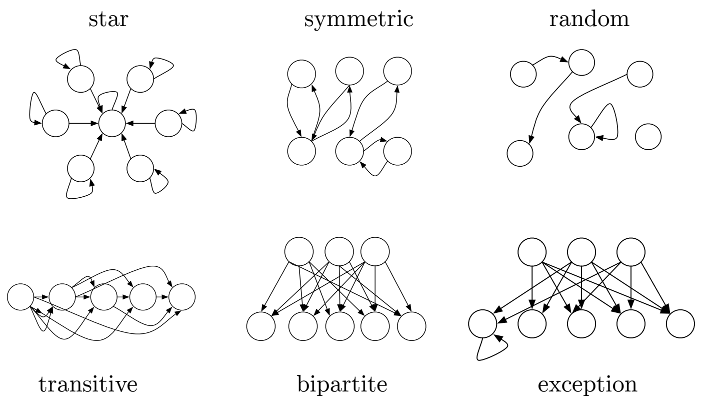
That said, at a computational level, the IKM can be used compute posterior probabilities of knowledge bases with negation-as-failure. To do so metarules with negation – e.g. $\exists ABC.A(X,Y) \leftarrow B(X), \neg C(Y)$ – are simply added to the knowledge base schema. And thus, using manually constructed knowledge bases, we can show that negation plays an important role in account for people's behavior when learning relational concepts. Kemp, Goodman, and Tenenbaum 2007 used an LOT model based on the relational knowledge bases to model how people learned six simple graph concepts in a laboratory setting. Figure 4-13 shows the negative log probability that our model assigns to a knowledge base for each one of these theories as a function as we vary whether the schema includes 1) binary facts or only unary facts or 2) negation. The presence of a binary fact is possible if the knowledge base schema



(a) Learning times and complexity ratings for subjects in Kemp, Goodman, and Tenenbaum 2007.



(b) Negative log probability of handwritten knowledge bases for each scenario.



(c) Graph concepts.

$$a(X,Y) \leftarrow \ell_1(X), \ell_2(Y)$$
$$\ell_1(1).\ell_1(2).\ell_1(3).\ell_1(4).\ell_1(5).$$
$$\ell_2(6).\ell_2(7).\ell_3(8).$$

(d) BIPARTITE knowledge without negation.

$$a(X,Y) \leftarrow \ell_1(X), \ell_2(Y).$$
$$\ell_1(X) \leftarrow obj(X), \neg\ell_2(X).$$
$$\ell_2(6).\ell_2(7).\ell_2(8).$$

(e) BIPARTITE knowledge with negation.

Figure 4-13: Comparing IKM to empirical data from Kemp, Goodman, and Tenenbaum 2007 with and without binary facts and negated atoms.

contains a rule of the form $\exists AXY.A(X,Y)$; if not a binary fact may be proven by a rule created with the metarule $\exists ABC.A(X,Y) \leftarrow B(X), C(Y)$ along with some unary facts. If we take the learning times and complexity ratings as proxies for the probability of the theories that subjects learn, we see that the IKM matches best when binary facts are not included and when negated atoms are used. The effect of negation is important because it plays an important role in the BIPARTITE and EXCEPTION theories (EXCEPTION is just BIPARTITE with some noise injected into it). Figures 4-13e and 4-13e show the manually written knowledge bases for the the BIPARTITE theory with and without negation, respectively. The use of negation allows for a much more compact knowledge base.

Full first-order logic vs. logic programming The model proposed in Kemp, Goodman, and Tenenbaum 2008 opted for the expressivity of full first-order logic (FOL). Our decision to use Datalog for our knowledge base formalism is motivated both by practical and theoretical considerations, and these considerations are largely the same as those that drove most practitioners in AI to search for restricted logics in which to express common-sense knowledge: FOL is undecidable, it is difficult to read and write, and it requires a automated theorem prover for reasoning (which are fragile, computationally demanding, and usually require interaction with an expert user). Additionally, FOL expressions can very easily be inconsistent – i.e. one can derive that some fact is both true and not true – and so modifying a knowledge base with FOL expressions means always checking the entirety for consistency. This is not the case in knowledge base formalisms, like Datalog, in which rules only tell you what is true, not what is false. For our purposes, which are focused on exploring algorithmic plausibility of Bayes LOT models, reasoning about a given expression in the LOT is just the inner loop in the search over expressions. This puts special pressure on the reasoning problem itself being relatively reliable and cheap.

Incremental hypothesis generation vs. stochastic local search We have discussed how inference IKM can be framed as state-space inference over a space of partial explanations of a data set, guided by a heuristic function. This is a very different approach from the majority of Bayes LOT approaches that use a form of stochastic local search to explore the hypothesis space. This dichotomy is a familiar AI, operations research, and combinatorial search, fields where there are vast and discrete space of possible solutions to a problem. Stochastic local search, which moving in a biased random walk, from one complete solution to another, is intuitive and easy to use. And, since it explores the hypothesis space broadly, it allows for approximate marginalization over the entire space of hypotheses (as opposed to MAP inference which only finds one or a few solutions).

But this approach is clearly not appropriate for many of the reasoning problems in which cognitive scientists are interested. People do not plan their actions by incrementally modifying a randomly chosen set of actions, and they do not write papers by proposing random strings of words until they happen upon one that means what they intend. It is plausible, then, that concept learning in language of thought also requires more than a stochastic search through the space of mental representations.

Why is concept learning possible for humans? When framed as a search problem over expressions in a language of thought, it becomes clear that concept learning over richly structured domains should be an extremely challenging problem. It is remarkable that it is possible at all. After all, as far as anyone can tell, no other animal is capable of acquiring such concepts. What can our concept learning approach here tell us, then, about why it is even possible for humans to acquire these concepts?

The work we have presented here suggests that the domain-generality of a learning approach – the degree to which it can be applied to disparate domains – is not equated with its simplicity. That is, just because a concept learning approach can be applied to any domain of problem does not necessarily mean that it must be in some sense "blind." The domain-general heuristic function employed by the IKM algorithm are complex, and this complexity does not emerge – as in Neural Networks – from a large number of uniform operations. Rather, IKM depends on the careful coordination of multiple data-structures and multiple search operators applied to a suitable representation language.

Therefore, it may not be implausible that evolution has only stumbled on such a coordination of computation components only once so far. On this view, what makes domain-general concept learning possible for humans and not for other animals is not any particular representation language or computational ability, but a suitable domain-general learning algorithm for searching the space of expressions in a language of thought.

Why is concept learning so hard? Why does it take so long? While concept learning is certainly possible for humans, it is clearly quite difficult. The case-study domains in this work take children many years to master. If the IKM is to be an account of human-like concept learning, what are we to make of the quickness with which it learns these concepts in these case-studies. Or, put otherwise, if the IKM can learn these concepts so quickly, why does it take children so long?

To some extent, the apparent speed of learning in these case-studies is the result of a methodological bias: we deliberately simplified the domains we studied. On the one hand, we want to use domains that are complex enough to capture the richness of the knowledge that children learn. But the development and study of learning algorithms would be severely limited if we could not quickly evaluate our techniques. Children learn a vast number of domains of knowledge at once, in a much more noisy environment than any of our case-studies capture. Heuristic search techniques make search faster by reducing the exponent on the runtime of algorithms whose run-times are otherwise exponential in size of the target solution and the branching factor of the search space; therefore, it should not be surprising that even with the best heuristics, even small increases in the complexity of the domain should lead to very large increases in search difficulty.

And yet this explanation is not quite satisfactory, for it does not explain why even though it takes children a very long time to learn, say, the meaning of the kinship concepts, that these concepts are eventually learned at a relatively young age is more or less certain. Notably, none of the models we discuss in this work have a pedagogical component; there is no feedback loop with a teacher that can guide the learner in any way. And while this may seem to be a reasonable elision for models of concept learning in very young children, it may be a critical omission when describing the progression of concept learning over the course of childhood.

Chapter 5

Afterword

In this thesis, we have presented two models of concept learning within the Bayes LOT framework, with the goal of tackling some of algorithmic challenges associated with learning in such models. First, we presented a model of learning as a search over the space of functional programs; we showed that our Exploration-Compression algorithm can learn and take advantage of the structure latent in a domain, reshaping the search space by discovering and caching useful recurring fragments. Our second model, the Infinite Knowledge Base Model (IKM), is a model of learning as a search over the space of first order relational knowledge bases. We framed learning in the IKM as a state space search over abductive proofs.

The technical core of this thesis has to do with automatically synthesizing programs. Finding methods for automatically synthesizing programs is a long sought after goal of the computational sciences. Despite major advances in search and optimization over the last century, it is safe to say that we have made little progress in understanding how to explore hypothesis spaces of structured and compositional expressions. Statisticians have algorithms to fit the parameters of their models, but first they must specify the structure of the model. Scientists run simulations to test their theories, but coming up with the theory is left to the scientists alone. In some cases, mathamaticians use theorem provers to help find proofs of their theorems, but they do not use them to suggest the theorems themselves. And none of this is for lack of effort.

If the Bayes LOT hypothesis is correct, as we believe it is, then automated program synthesis is a fundamental problem for cognitive science as well. Our goal in this thesis has been to move us in that direction, by a) specifying Bayes LOT models that expose the vastness of the hypothesis space with which every child is faced and by b) exploring algorithmic principles that take advantage of the the structured and compositional character of these representation languages.

As discussed throughout this thesis, much of the previous work on the Bayes LOT hypothesis has been at David Marr's *computational* level of analysis: it has argued that the human learner effectively behaves as rational Bayesian learner who faces a prior distribution over expressions in a language of thought and who wishes to understand the world in terms of those expressions. As this thesis demonstrates, going beyond this level of analysis to the *algorithmic* level is challenging. But even if we are not able to describe a practicle or realizable algorithm for learning in a Bayes LOT model, understanding how such an algorithm *might* work is important for cognitive science.

To make an analogy to another branch of the natural sciences: although we are unable to simulate in our computers the evolution of organisms on any significant scale, our current understanding of evolution goes well beyond the "computational" level statement that organisms evolve to optimize their fitness. We understand it at an "algorithmic" level; for example, we know that evolution is driven by genetic recombination and random mutations. This allows us to understand natural phenomena (like speciation) and guides our scientific explorations of the "implementation" level, whether in terms of mating behaviors or the structure of the genome.

We hope that, similarly, a better understanding of the principles of learning in Bayes LOT models provides us with insight into such diverse issues as the peculiar dynamics of learning, the nature of pedagogy and the role of language in cognition. For we think that this work makes clear that, at least for humans, learning cannot be ascribed a passive metaphore in the mind; it cannot be *merely* the consolidation of memories or the tuning of neural weights or the pruning of connections. Any mechanism that is capable of effectively exploring expressions in the language of thought is unlikely to leave any other aspect of mental life unaffected.

Appendix A

EC as variational inference in a hierarchical probabilistic model over functional programs

In this section, we show how the EC algorithm can be seen as implementing an approximate variational inference algorithm in a hierarchical probabilistic model over functional programs. The probabilistic model we employ is largely the same as the one presented in Liang, Jordan, and Klein 2010. In particular, expressions are assumed to be generated from an *adaptor grammar* (Johnson, Griffiths, Goldwater, et al. 2007). An adaptor grammar is a stochastic generative process over trees $T_1, T_2, \ldots$ with the property that subtrees generated in the past are more likely to be generated in the future. Thus, like many non-parametric Bayesian processes based on the Dirichlet process, they exhibit a "rich gets richer" dynamic. One way to think of an adaptor grammar is as a PCFG that stochastically memoizes the subtrees it generates: each time a subtree is generated, there is some probability that this subtree is added to the grammar as an atomic production. And the probability of using a previously cached subtree increases the more it has been previously used.

The adaptor grammar implements precisely the intuition that motivates our approach to multitask program synthesis: in the context of generating expressions in a programming language, the programmer's library functions is her finite approximation to the most high probability elements of the adaptor grammar's latent collection of cached subtrees.

Liang, Jordan, and Klein 2010 propose a Metropolis-Hastings algorithm for an adaptor grammar over functional programs. This sampling-based approach relies on the previously described representation of adaptor grammar as a process that sequentially generates programs and stochastically reusing frequently used subexpressions. In this representation, which we will call the *sequential representation* of the adaptor grammar, there is an explicit dependency between previously generated trees and those generated in the future. However, as shown in Cohen, Blei, and Smith 2010, there is also non-sequential *stick-breaking* representation of the adaptor grammar that makes these dependencies implicit by introducing a latent (infinitely large) collection of expressions conditioned on which the T_i are i.i.d. The stick-breaking representation makes an iterative variational inference approach feasible.

Variational inference is a general framework for approximate inference in probabilistic models. The general setting is that the learner wants to estimate $p(z|x)$, the conditional distribution of some latent variables z given some observed data x, but only has access to the joint distribution $p(x, z)$. In the common case that computing $p(z|x)$ directly in not feasible, the learner attempts to find a distribution $\hat{q} = q(\hat{\phi})$ from among the family of distributions Q which is "close" to $p(z|x$. Specifically, the goal is to find $\hat{q} = \arg\min_{q\in Q} \mathrm{KL}(q(z)||p(z|x))$, where $KL(\cdot||\cdot)$ is the KL divergence. The *mean-field* approximation is a very common approach to variational inference and is obtained by specifying Q as a product of independent distributions where $q(z; \phi_1, \ldots, \phi_F) = q_1(z_1; \phi_1) \ldots q_F(a_F; \phi_F)$ and $z = z_1 \cup \cdots \cup z_F$. Specifically, we can use the general mean-field update equations:

$$q_j(\phi_j) \leftarrow \frac{\exp \mathbb{E}_{q_{i\neq j}} [\log p(x, z)]}{\int d\phi_j \exp \mathbb{E}_{q_{i\neq j}} [\log p(x, z)]}. \tag{A.1}$$

Procedurally, mean-field variational inference can be implemented by cycling through the factors of the approximating the distribution and, for each factor, estimating its parameter values by taking an expectation over all the other factors, using Equation A.1.

Let A be an adaptor grammar with base grammar $\mathbf{G}_0$, adapted non-terminals M, discount parameter b, concentration parameter a and a set of Dirichlet parameters α. The base grammar $\mathbf{G}_0$ is a context free grammar with terminals W, nonterminals N, production rules R, and a start symbol S. The set of rules with left hand side non-terminal A will be denoted R_A.

In the non-sequential representation, derivation trees z for expressions are sampled as follows. For each adapted nonterminal $A \in M$, we sample the vector $\pi_A = \pi_{A,1}, \pi_{A,2}, \ldots$ where $\sum_i \pi_{A,i} = 1$ from $GEM(a_A, b_A)$. This vector is then used to define a categorical distribution over expressions $u_A = u_{A,1}, u_{A,2}, \ldots$ where each $u_{A,1}$ is itself sampled from the adaptor grammar; see 7. Finally, the derivation trees for each task $z_1, \ldots, z_N$ are sampled by additional independent calls to the adaptor grammar.

Algorithm 7: Generative process for sampling derivation trees from adaptor grammar based on stick-breaking construction of GEM process

```
Def AgGenTree($A$):
    draw $A \rightarrow B_1 \ldots, B_N$ from $R_A$ with probability $\theta_A$
    $z \leftarrow A \Longrightarrow B_1 \ldots, B_N$
    while $yield(z)$ contains nonterminals do
        $B \leftarrow$ unexpanded nonterminal in $z$
        if $B \in M$ then
            replace $B$ with a draw from $G_A$
        else
            expand $B$ from $R_B$ with probabilities $\theta_B$
        end
    end
    return $z$
Def AgGenGrammar($A$):
    $\pi | a_A, b_A \sim \text{GEM}(a_A, b_B)$
    for $i \in 1, \ldots$ do
        $z_i \leftarrow$ AgGenTree($A$)
    end
    return Categorical($\{z_1, \ldots\}, \bar{\pi}$)
Def AgGenGrammar:
    for $A \in M$ do
        $\mathbf{G}_A \leftarrow$ AgGenGrammar($A$)
    end
    return $\mathbf{G}_A$ for $A \in M$
Def AgGenExpressions:
    for $i \in 1, \ldots, N$ do
        if $S \in M$ then
            $z_i \sim \mathbf{G}_S$
        else
            $z_i \sim$ AgGenTree($S$)
        end
    end
    return $\{z_1, \ldots, z_N\}$
```

In the context of learning programs, the variables of interest π, θ, u and z. Instead of directly dealing with the normalized vector π_A, it is more convenient to deal with the unnormalized proportions v_A, where $\pi_{A,i} = v_{A,i} \prod_{j<i}(1 - v_{A,j})$ because each $v_{A,j}$ is independently distributed from the Beta distribution $\text{Beta}(1 + a_A, a + ib)$. To generate a mean-field variational distribution over these variables, we describe the space of variational approximations Q as the product of independent distributions over these variables:

$$Q = \{ \prod_A q_\theta(\theta_A; \tau_A) \prod_i q_v(v_{A,i}|\gamma^{(1)}_{A,i}, \gamma^{(2)}_{A,i}) \prod_j q_u(u_j|\phi^{(u)}) \prod_j q_z(z_j|\phi^{(z)})| \tag{A.2}$$

$$q_\theta(\cdot|\tau) \sim \text{Dirichlet}(\tau) \tag{A.3}$$

$$q_v(\cdot|\gamma^{(1)}, \gamma^{(2)}) \sim \text{Beta}(\gamma^{(1)}, \gamma^{(2)}) \tag{A.4}$$

$$q_u(\cdot|\phi) \sim \text{PCFG}(\Delta, \phi) \tag{A.5}$$

$$q_z(\cdot|\phi) \sim \text{PCFG}(\Delta, \phi)\}. \tag{A.6}$$

Given this family of approximating distributions, we derive the variational update equations using Equation A.1, where the target joint probability distribution $p = p(t, z, u, \pi, \theta; a, b, \alpha)$.

The distributions q_z and q_u are drawn from a common PCFG G' with production probabilities ϕ and production rules R', and for this distribution, Johnson et al 2005 and Cohen et al 2010 use a set of rules derived in a bottom-up fashion from the string or strings under consideration. In our case, however, there is no set of strings from which these PCFGs can be derived. Let R_A be the set of rules in R with head nonterminal A. Then, we specify the rules

$$R'_A = R_A \cup \{A \rightarrow \beta_1 \dots \beta_N | A \leftarrow B_1 \dots B_N, \tag{A.7}$$

$$\beta_1 \in \text{YIELD}(B_1), \dots, \beta_N \in \text{YIELD}(B_N)\} \tag{A.8}$$

Where define the yield of a nonterminal to be all possible derivation trees that are rooted at that nonterminal. So R'_A contains all the rules headed by nonterminal A in R_A plus the potentially infinite set of rules whose right hand side is a string of terminals that could be generated by the right hand side of some rule in R_A.

When computing the expected number of time the rule $A \rightarrow u_i$ is used in some derivation z, $f(A \rightarrow u_i, z)$, we must compute an approximate expectation, which we do by enumerating a finite number of derivations. However, for any task whose solutions are of a nontrivial length, we must do this in such a way that there is a good probability that we will actually find expressions that solve our task. The approach that EC takes is to perform a best-first search on the distribution over parse trees imposed by current variational approximation. If we also wish to have cached expressions of non-trivial size, then we must allow both the parameters and the production rules themselves to change as the approximation continues. Otherwise, to use a cached subexpression containing N nodes, we would need to store $O(\exp(N))$ production rules in our approximate PCFG.

Suppose that at some iteration k in our variational inference procedure we have an approximate PCFG with rules Δ and production probabilities Φ. Let $\text{BF}(N, \Delta, \Phi)$ be the N derivation trees returned by the best-first search of derivations trees. Then for each derivation tree $z \in \text{BF}(N, \Delta, \Phi)$, we can compute the approximate expected counts $f(u_A)$ of cached expression u_A, rooted at adapted nonterminal A. Instead of considering only those derivation trees produced by the best-first enumeration, we want to consider expressions that are not in our approximate grammar as well. Let

$U_A = \mathrm{BF}(N, \Delta, \Phi)$ contain any span that can be rooted by adapted nonterminal A and that is not a right hand side for some rule in Δ and is contained in some derivation in $\mathrm{BF}(N, \Delta, \Phi)$. Let Δ' be a set of PCFG rules augmented with this set of U:

$$\Delta' \triangleq \Delta \cup \{A \to u | A \in M, u \in U\}. \tag{A.9}$$

To initialize the corresponding production probabilities Φ' for the new rules, we extend γ^1 and γ^2 for the corresponding expressions and recompute the Φ' parameters accordingly.

$$\gamma^{1'} = f_{\gamma^1}([f(A \to u_A)|0, \overset{|U|}{\ldots}, 0] \tag{A.10}$$
$$\gamma^{2'} = f_{\gamma^2}([f(A \to u_A)|0, \overset{|U|}{\ldots}, 0] \tag{A.11}$$
$$\phi' = f_\phi(\gamma^{1'}, \gamma^{2'}). \tag{A.12}$$

Now we can compute the expected rule counts using this augmented grammar. There are two components to the expected rule counts. For a given rule r, we want to count the expected number of times it used in generating the tasks t and the expected number of times it is used in generating the cached subexpressions u. The former we can do using the Inside-Outside algorithm.

The latter distribution, q_u, is not conditioned on the tasks, so we just compute the unconditional expectation over the K independently drawn subexpressions for each adapted nonterminal $A \in M$. Let c be a matrix where $c_{A,r}$ is the expected number of occurrences of rule r in a derivation tree with root A. Let $\phi_{A,r}$ be the probability of expanding a nonterminal A using rule r and $N_{r,A}$ be the number of times nonterminal A occurs on the right hand side of rule r. Then c is solution to $\phi = c(1 - (\phi N)^T)$.

The adaptor grammar inference procedure places pressure on the estimated distribution in two ways. First, larger cached subexpressions are preferable to shorter ones: consider an expression e corresponding to a binary tree of all of whose subtrees are distinct (e.g. the leaves are numbered $1, 2, \ldots$), and suppose that we are computing the expected counts of rules used to derive this expression where the rules are $\{S \to (S\,S)\} \cup \{S \to u | u \text{ is a subexpression of} e\}$ and all rule probabilities are equal. Then the rule count rule $S \to u_n$ where u_n is a subexpression of height n will have twice the expected count of the rule $S \to u_{n-1}$. The second pressure comes from the GEM prior over stick-breaking proportions: the prior accentuates the differences between subexpressions. These two tendencies together mean that the approximate posterior will try to put its weight on large rules that occur frequently, and that, where, there are multiple similar rules, the posterior will concentrate its mass on one or a few of these rules, rather than keep it spread out.

Algorithm 8: EC as mean-field variational inference over an Adaptor Grammar

```
Data: $R$: rules; $R_A$ rules with left hand side $A$
$N$: nonterminals; $M$: adapted nonterminals
$a$; $b$; $\alpha$
$N_F$: frontier size
$\triangleright$ initialization
for $A \in N$ do
    $\Delta_A \leftarrow R_A$
end
initialize $f$ to zeros
$\phi \leftarrow g_\tau(r; f, \Delta)$
$\triangleright$ loop
while not converged do
    $F \leftarrow \mathrm{BF}(N_f, \Delta, \phi)$ $\triangleright$ explore frontier of $N_F$ expressions
    for task $z_i$ do
        $F_i \leftarrow \{e | e \in F, z_i(e)\}$
    end
    $F \leftarrow \cup_i F_i$
    for every subexpression $u$ of an expression $e \in F$ do
        insert $A \rightarrow u$ into $\Delta_A$ at smallest unoccupied index
    end
    $\phi \leftarrow g(f, \Delta)$ $\triangleright$ compute $\phi$ for new rules
    for $r \in \Delta$ do
        $f_r \leftarrow \mathbb{E}_F[r | \Delta, \phi]$ $\triangleright$ compute expected rule counts
    end
    $\phi \leftarrow g(f, \Delta)$ $\triangleright$ compute new production probabilities
    for $A \in M$ do
        $\Delta_A \leftarrow$ first $K$ $r \in \Delta_A$ sorted in descending order of $\phi_r$
    end
end
```

$$g_{\gamma^1}(r = A \to \_; f, \Delta) = 1 - b_A + \sum_{r \in \Delta_A} f_r \tag{A.13}$$

$$g_{\gamma^2}(r = A \to \_; f, \Delta) = a_A + i_r b_A + \sum_{r' \in \Delta_A} 1(i'_r < i_r) f'_r \tag{A.14}$$

$$\text{where } i_r = \text{index of rule } r \text{ in } \Delta_A \tag{A.15}$$

$$g_\tau(r; f) = \alpha_A + f_r \tag{A.16}$$

$$g_\phi(r = A \to \_; \Delta, \gamma^1, \gamma^2, \tau) = \begin{cases} \Psi(\gamma^1_r) - \Psi(\gamma^1_r + \gamma^2_r) + \sum_{r' \in \Delta_A} (\Psi(\gamma^2_{r'}) - \Psi(\gamma^1_{r'} + \gamma^2_{r'})), \text{ if } A \in M \\ \Psi(\tau_r) - \Psi(\sum_{r' \in \Delta_A} \tau'_r), \text{ otherwise} \end{cases} \tag{A.17}$$

$$g(r; f, \Delta) = g_\phi(r; \Delta, g_{\gamma^1}(f, \Delta), g_{\gamma^2}(f, \Delta), g_\tau(f)) \tag{A.18}$$

Figure A-1: Update functions for mean-field variational inference

$$q_z(z|t, \Delta', \phi', \Delta, \phi) \propto 1(e \in \text{BF}(N, \Delta, \phi)) 1(\text{yield}(z) = e) 1(t(e)) q_z(z|\Delta', \phi') \tag{A.19}$$

$$f(A \to u|\Delta, \phi) = \sum_{t,z} \sum_{A \to u \in z} q_z(z|t, \Delta', \phi', \Delta, \phi) \tag{A.20}$$

$$+ \sum_{B \in M} \sum_{B \to u' \in \Delta'} \mathbb{E}_{\substack{r \in z : \\ \text{yield}(z) = u'}} [1(r = A \to u)|\Delta', \phi'] \tag{A.21}$$

$$\tag{A.22}$$

Finally, we compute new γ parameters from these expected counts, which we then sort and truncate to T values. That gives us the γ values for the next iteration.

Additionally, we need to decide which rules in R'_A will correspond to the parameters $\gamma_{A,i}$. We do this according to the heuristic that larger stick proportions should be assigned to the rules that have the largest expected counts. This leads us to the following procedure: a) compute the expected counts f_{r_i} for rules r_i, b) set $\gamma_{A,1}, \ldots, \gamma_{A,K} = \text{SORT}\{f_{r_1}, \ldots, f_{r_K}\}$. Note that we compute the expected counts for some rule f_r for every r that appears in our enumeration of derivations, and not only those r's that already have a variational parameter assigned to them. But for any rule r that is not included in the variational parameters on iteration j, we assign the default variational parameters $\gamma^{(1)}_r = 1 - b_A$ and $\gamma^{(2)}_r = a_{h(r)} + K b_{h(r)} + \sum$.

To apply mean-field variational inference when inferring these variables, we factorize as these latent variables as

$$q(\theta, \{z_{A,i}\}_{A,i}, \pi_A, \{z_i\}_i) = q_{\text{tasks}}(\{z_i\}_i) q_{\mathbf{G}}(\{z_{A,i}\}_{A,i}) q_\pi(\pi) q_\theta(\theta) \tag{A.23}$$

where q_{tasks} and $q_{\mathbf{G}}(\{z_{A,i}\}_{A,i})$ are categorical distributions over parse trees and q_π and q_θ Dirichlet distributions.

In practice, it is necessary to limit the cached parse trees $z_{A,i}$ to a finite, and preferably small, number.

A.1 Circuit Distribution

The procedure for sampling circuits for Experiment 1 of Section 2.6.1 is as follows:

1. sample the number of Boolean inputs N according to discrete distribution

N	1	2	3	4
$p(N) \propto$	1	2	4	4

2. sample the number of gates M according to discrete distribution

M	1	2	3	4	5	6
$p(M) \propto$	1	3	4	4	5	5

3. the output of the Mth gate is considered the output of the circuit

4. until there are not unconnected inputs in the circuit, connect an arbitrarily selected output to an unconnected input

Bibliography

Abiteboul, Serge, Richard Hull, and Victor Vianu (1995). *Foundations of databases: the logical level.* Addison-Wesley Longman Publishing Co., Inc.

Adams, Ryan Prescott, Hanna M. Wallach, and Zoubin Ghahramani (2010). "Learning the Structure of Deep Sparse Graphical Models". In: *Journal of Machine Learning Research: Workshop and Conference Proceedings (AISTATS)* 9, pp. 1–8.

Booth, Taylor L and Richard A Thompson (1973). "Applying probability measures to abstract languages". In: *IEEE transactions on Computers* 100.5, pp. 442–450.

Briggs, Forrest and Melissa O'Neill (2008). "Functional genetic programming and exhaustive program search with combinator expressions". In: *KES Journal* 12.1, pp. 47–68.

Carey, Susan (1985). "Conceptual change in childhood". In:

Cohen, Shay B, David M Blei, and Noah A Smith (2010). "Variational inference for adaptor grammars". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics.* Association for Computational Linguistics, pp. 564–572.

Curry, Haskell B. and Robert Feys (1968). *Combinatory Logic.* 2nd. Vol. 1. Studies in Logic and the Foundations of Mathematics. Amsterdam: North-Holland.

Dale, Philip S and Larry Fenson (1996). "Lexical development norms for young children". In: *Behavior Research Methods, Instruments, & Computers* 28.1, pp. 125–127.

Dechter, Eyal and J. B. Tenenbaum (in prep.). "The Infinite Knowledge Base Model: A search-based approach to probabilistic inference over first-order knowledge bases". In: In prep.

Dechter, Eyal et al. (2013). "Bootstrap Learning via Modular Concept Discovery". In: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013.* Ed. by Francesca Rossi. IJCAI/AAAI.

Dechter, Rina (2003). *Constraint processing.* Morgan Kaufmann.

Dempster, Arthur P, Nan M Laird, and Donald B Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38.

Edelkamp, Stefan and Stefan Schrödl (2012). *Heuristic Search - Theory and Applications.* Academic Press, $\mathrm{I}_X XIV$, 1˘836.

Ellis, Kevin, Eyal Dechter, and Joshua B Tenenbaum (2015). "Dimensionality Reduction via Program Induction". In: *2015 AAAI Spring Symposium Series.*

Ellis, Kevin et al. (2013). “Learning Graphical Concepts”. In: *NIPS Workshop on Constructive Machine Learning.*

Fikes, Richard E, Peter E Hart, and Nils J Nilsson (1972). “Learning and executing generalized robot plans”. In: *Artificial intelligence* 3, pp. 251–288.

Fodor, Jerry A. (1975). *The Language of Thought.* New York: Thomas Y. Crowell Company, Inc.

Fodor, Jerry A and Zenon W Pylyshyn (1988). “Connectionism and cognitive architecture: A critical analysis”. In: *Cognition* 28.1, pp. 3–71.

Friedman, N. and D. Koller (2003). “Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks”. In: *Machine Learning* 50.1-2, pp. 95–125.

Gabbay, Dov M, Christopher John Hogger, and John Alan Robinson (1998). *Handbook of Logic in Artificial Intelligence and Logic Programming: Volume 5: Logic Programming.* Clarendon Press.

Geffner, Hector and Blai Bonet (2013). *A Concise Introduction to Models and Methods for Automated Planning.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, pp. 1–141.

Gelman, Andrew et al. (2014). *Bayesian data analysis.* Vol. 2. Chapman & Hall/CRC Boca Raton, FL, USA.

Gent, Ian et al. (1996). “An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem”. In: *Principles and Practice of Constraint Programming—CP96.* Springer, pp. 179–193.

Goodman, Noah D et al. (2008a). “A Rational Analysis of Rule-Based Concept Learning”. In: *Cognitive Science* 32.1, pp. 108–154.

Goodman, Noah D et al. (2008b). “Compositionality in rational analysis: Grammar-based induction for concept learning”. In: *The probabilistic mind: Prospects for Bayesian cognitive science.*

Gopnik, Alison, Andrew N Meltzoff, and Peter Bryant (1997). *Words, thoughts, and theories.* Vol. 1. Mit Press Cambridge, MA.

Gulwani, Sumit (2011). “Automating string processing in spreadsheets using input-output examples”. In: *ACM SIGPLAN Notices.* Vol. 46. 1. ACM, pp. 317–330.

Hart, Peter E, Nils J Nilsson, and Bertram Raphael (1968). “A formal basis for the heuristic determination of minimum cost paths”. In: *IEEE transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

Hauskrecht, Milos et al. (1998). “Hierarchical solution of Markov decision processes using macro-actions”. In: *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence.* Morgan Kaufmann Publishers Inc., pp. 220–229.

Heckerman, David, Dan Geiger, and David Maxwell Chickering (2013). “Learning Bayesian Networks: The Combination of Knowledge and Statistical Data”. In: *CoRR* abs/1302.6815.

Johnson, Mark, Thomas L Griffiths, Sharon Goldwater, et al. (2007). “Adaptor grammars: A framework for specifying compositional nonparametric Bayesian models”. In: *Advances in neural information processing systems* 19, p. 641.

Katayama, Susumu (2005). “Systematic search for lambda expressions.” In: *Trends in functional programming* 6, pp. 111–126.

Katz, Yarden et al. (2008). "Modeling semantic cognition as logical dimensionality reduction". In: *Proceedings of thirtieth annual meeting of the cognitive science society.*

Kemp, Charles, Noah Goodman, and Joshua B Tenenbaum (2007). "Learning and using relational theories". In: *Advances in neural information processing systems*, pp. 753–760.

Kemp, Charles, Noah D Goodman, and Joshua B Tenenbaum (2008). "Theory acquisition and the language of thought". In: *Proceedings of the 30th annual conference of the Cognitive Science Society.* Cognitive Science Society.

Kemp, Charles et al. (2006). "Learning systems of concepts with an infinite relational model". In: *AAAI.* Vol. 3, p. 5.

Khan, Khalid, Stephen Muggleton, and Rupert Parson (1998). "Repeat Learning Using Predicate Invention". In: *ILP*, pp. 165–174.

Korf, Richard E. (1985a). "Depth-First Iterative-Deepening: An Optimal Admissible Tree Search". In: *Artif. Intell.* 27.1, pp. 97–109.

— (1985b). "Macro-Operators: A Weak Method for Learning." In: *Artif. Intell.* 26.1, pp. 35–77.

Koza, John R. (1993). *Genetic programming - on the programming of computers by means of natural selection.* Complex adaptive systems. MIT Press, pp. I–XVIII, 1–419.

Krishnan, Thriyambakam and G McLachlan (1997). "The EM algorithm and extensions". In: *Wiley* 1.997, pp. 58–60.

Leslie, Alan M, Rochel Gelman, and CR Gallistel (2008). "The generative basis of natural number concepts". In: *Trends in cognitive sciences* 12.6, pp. 213–218.

Liang, Percy, Michael I. Jordan, and Dan Klein (2010). "Learning Programs: A Hierarchical Bayesian Approach". In: *ICML*, pp. 639–646.

Manning, Christopher D, Hinrich Schütze, et al. (1999). *Foundations of statistical natural language processing.* Vol. 999. MIT Press.

Marr, David (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* New York, NY, USA: Henry Holt and Co., Inc.

Miller, Dale and Gopalan Nadathur (2012). *Programming with Higher-Order Logic.* Cambridge University Press.

Muggleton, Stephen (1987). "Duce, An Oracle-based Approach to Constructive Induction." In: *IJCAI.* Citeseer, pp. 287–292.

— (1997). "Learning from positive data". In: *Inductive logic programming*, pp. 358–376.

Muggleton, Stephen and Luc De Raedt (1994). "Inductive Logic Programming: Theory and Methods". In: *J. Log. Program.* 19/20, pp. 629–679.

Muggleton, Stephen et al. (2012). "ILP turns 20 - Biography and future challenges". In: *Machine Learning* 86.1, pp. 3–23.

Muggleton, Stephen H. and Dianhuan Lin (2013). "Meta-Interpretive Learning of Higher-Order Dyadic Datalog: Predicate Invention revisited". In: *IJCAI.* Ed. by Francesca Rossi. IJCAI/AAAI.

Nevill-Manning, Craig G. and Ian H. Witten (1997). “Identifying Hierarchical Structure in Sequences: A Linear-Time Algorithm”. In: *J. Artif. Intell. Res. (JAIR)* 7, pp. 67–82.

O’Donnell, Timothy J (2015). *Productivity and reuse in language: A theory of linguistic computation and storage.* MIT Press.

Pearl, Judea (1984). *Heuristics - intelligent search strategies for computer problem solving.* Addison-Wesley series in artificial intelligence. Addison-Wesley, pp. I–XVII, 1–382.

— (1989). *Probabilistic reasoning in intelligent systems - networks of plausible inference.* Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, pp. I–XIX, 1–552.

Piantadosi, S.T., J.B. Tenenbaum, and N.D. Goodman (2010). “Beyond Boolean logic: exploring representation languages for learning complex concepts”. In: *Proceedings of the 32nd Annual Conference of the Cognitive Science Society.*

Piantadosi, S.T., J.B. Tenenbaum, and N.D Goodman (2012). “Bootstrapping in a language of thought: a formal model of numerical concept learning”. In: *Cognition* 123, pp. 199–217.

Pierce, Benjamin C. (2002). *Types and programming languages.* MIT Press, pp. I–XXI, 1–623.

Pitman, J. (2006). *Combinatorial Stochastic Processes: Ecole d’Eté de Probabilités de Saint-Flour XXXII - 2002.* Ed. by J. Picard. Lecture Notes in Mathematics. Springer Berlin Heidelberg.

Randlov, Jette (1999). “Learning macro-actions in reinforcement learning”. In: *Advances in Neural Information Processing Systems*, pp. 1045–1051.

Rendell, Larry (1985). *Substantial constructive induction using layered information compression: Tractable feature formation in search.* Department of Computer Science, University of Illinois at Urbana-Champaign.

Russell, Stuart J. and Peter Norvig (2010). *Artificial Intelligence - A Modern Approach (3. internat. ed.)* Pearson Education.

Sarnecka, Barbara W and Susan Carey (2008). “How counting represents number: What children must learn and when they learn it”. In: *Cognition* 108.3, pp. 662–674.

Schönfinkel, Moses (1967). “On the building blocks of mathematical logic”. In: *From Frege to Gödel.* Ed. by J. van Heijenoort. Harvard University Press, pp. 355–366.

Schulz, LE (2012). “Finding new facts; thinking new thoughts”. In: *Rational constructivism, Advances in child development and behavior* 42.

Simon, Herbert A (1996). *The sciences of the artificial.* MIT press.

Spelke, Elizabeth S and Katherine D Kinzler (2007). “Core knowledge”. In: *Developmental science* 10.1, pp. 89–96.

Tenenbaum, Joshua B. (1999). “Bayesian Modeling of Human Concept Learning”. In: *Advances in Neural Information Processing Systems 11.* Ed. by M. J. Kearns, S. A. Solla, and D. A. Cohn. MIT Press, pp. 59–68.

Tsamardinos, Ioannis, Laura E. Brown, and Constantin F. Aliferis (2006). “The max-min hill-climbing Bayesian network structure learning algorithm”. In: *Machine Learning* 65.1, 31–78.

Ullman, T. D., N. D. Goodman, and J. B. Tenenbaum (2012). “Theory learning as stochastic search in the language of thought”. In: *Cognitive Development*.

Wellman, Henry M and Susan A Gelman (1992). “Cognitive development: Foundational theories of core domains”. In: *Annual review of psychology* 43.1, pp. 337–375.

Winograd, Terry (1972). “Understanding natural language”. In: *Cognitive psychology* 3.1, pp. 1–191.

Wynn, Karen (1992). “Children’s acquisition of the number words and the counting system”. In: *Cognitive Psychology* 24.2, pp. 220 –251.

Yakushev, Alexey Rodriguez and Johan Jeuring (2009). “Enumerating Well-Typed Terms Generically”. In: *Approaches and Applications of Inductive Programming, Third International Workshop, AAIP 2009, Edinburgh, UK, September 4, 2009. Revised Papers*, pp. 93–116.